Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Trabalho de Formatura Supervisionado

Vitruvius

Um Reconhecedor de Gestos para o Kinect™

Tiago Andrade Togores

Orientador: Professor Flávio Soares Corrêa da Silva

1 de dezembro de 2011

Agradecimentos

Agradeço à empresa Opus Software pelo tempo em que parte do trabalho aqui apresentado foi realizado e pelas experiências adquiridas com o desenvolvimento de aplicações com o framework .NET e para o Kinect. Agradeço também ao Professor Flávio Soares Corrêa da Silva pelas diretrizes e conselhos oferecidos ao longo do ano. E, por último mas não menos importante, à minha família que contribuiu para os experimentos práticos realizados.

Resumo

Interfaces naturais são modos de interação homem-máquina cada vez mais populares. Permitem que as pessoas se comuniquem com dispositivos de modo intuitivo, mas do outro lado precisa haver um software que entenda a linguagem.

Este trabalho consiste numa aplicação desenvolvida usando o Kinect, que aprende gestos com as mãos executados por usuários para posteriormente fazer o reconhecimento deles. Usamos uma sequência de processamento formada por extração de vetores de características, filtagem, quantização e Modelo Oculto de Markov (HMM).

Uma série de testes foi realizada de modo a obter a maior acurácia possível do sistema. Os resultados e a análise são apresentados ao final dessa monografia.

Sumário

Prefácio

Esta monografia foi concebida na disciplina Trabalho de Formatura Supervisionado (MAC0499), cursada pelo graduandos do curso de Bacharelado em Ciência da Computação (BCC), do Departamento de Ciência da Computação (DCC), do Instituto de Matemática e Estatística (IME), da Universidade de São Paulo (USP), e ministrada no ano de 2011 pelo Professor Carlos Eduardo Ferreira (~cef).

Todos os produtos deste trabalho (apresentação, pôster, código-fonte e etc.) podem ser encontrados no sítio:

`http://www.linux.ime.usp.br/~togores/mac499`

O conteúdo do documento está dividido em duas partes. Parte objetiva:

Seção 1
: Contextualização, motivação, apresentação do problema e da solução proposta.

Seção 2
: Descrição das tecnologias utilizadas.

Seção 3
: Definição de conceitos técnicos e matemáticos. Exposição e explicação dos algoritmos utilizados.

Seção 4
: O que e como foi desenvolvido no período, incluindo problemas encontrados, mudanças de planos e metodologias.

Seção 5
: Análise e comparação dos resultados.

Seção 6
: Considerações finais.

Parte subjetiva:

Seção 1
: Desafios e frustrações encontrados durante o ano.

Seção 2

Disciplinas cursadas mais relevantes para o trabalho. Quais os conceitos que foram aplicados.

Seção 3

Qual o caminho a ser seguido.

Parte I
Objetiva

1 Introdução

1.1 Contextualização

Ao longo da história, foram desenvolvidos diversos modos de interação humano-computador, porém sempre com o mesmo objetivo: tornar o controle da máquina pelo homem eficiente, de modo que os resultados desejados fossem obtidos. Entretanto, apenas este aspecto não é suficiente para que o usuário tenha uma experiência agradável.

Interfaces naturais permitem que usuários se comuniquem com dispositivos de forma mais intuitiva, com uma curva de aprendizado baixa. Em pouco tempo, a interface fica praticamente imperceptível e a comunicação se torna natural.

Este conceito foi mais disseminado recentemente com a acessibilidade crescente de tecnologias como superfícies multitoque em smartphones e tablets pelo público geral. Anteriormente, já havia o reconhecimento de voz e fala. Em 2010, o Kinect, um periférico originalmente desenvolvido para o console Xbox 360 pela Microsoft, passou a permitir o desenvolvimento de programas que realizam comunicação através de tanto gestos corporais quanto som. Logo após seu lançamento, a comunidade de software livre conseguiu adaptar seu funcionamento para computadores pessoais.

No entanto, uma séria limitação existente para o usuário final é que as aplicações criadas até hoje somente reconhecem um conjunto fixo de gestos ou permitem a adição de gestos apenas simples. Comparação da soma das distâncias de amostras de pontos com um certo limite estão presentes em boa parte dos algoritmos, assim como a observação se uma coordenada cartesiana do movimento incrementou ou não. Nestes casos, a complexidade de descrição de um gesto aumenta proporcionalmente com a do próprio. Além disso, eles não tratam bem as sutis modificações na execução inerentes ao ser humano.

1.2 Objetivo

Uma das melhores maneiras de contornar esse problema é aplicando conhecimentos da área de aprendizado computacional. Cada gesto seria associado a um modelo estatístico, que devolve uma probabilidade de que um gesto feito seja o que ele (modelo) representa. Ademais, cada modelo é treinado de modo que descreva com maior precisão a ação desejada. Tem-se, assim, duas vantagens:

- Qualquer um pode criar seus próprios gestos;

- Os gestos virtualmente não tem limite de complexidade;

O trabalho desenvolvido consiste, então, em um aplicativo que permite treinamento e reconhecimento de gestos de modo prático para o usuário final, utilizando o Kinect. Para tanto, é necessário o estudo do funcionamento do dispositivo e de suas aplicações e a implementação de uma técnica de inteligência artifical adequada.

2 Tecnologias

2.1 Kinect

É um periférico vendido pela Microsoft para uso no XBOX 360, um videogame, que permite ao usuário interagir de forma natural com o console, através de gestos corporais e comandos de voz, com a finalidade de entretenimento. Seus recursos são:

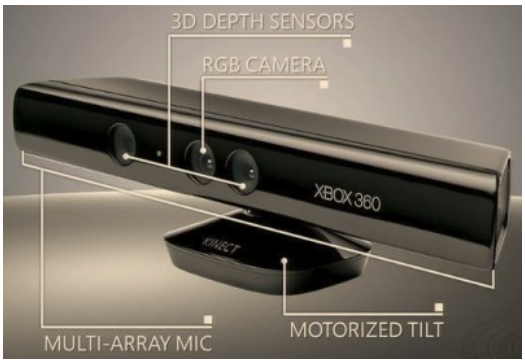

Figura 1: Kinect

- Câmera RGB: obtém imagens coloridas numa resolução de 640 por 480, com uma taxa de atualização de 30 quadros por segundo. Cada pixel é representado por 32 bits, sendo que apenas 24 deles são usados, com cada 8 determinando o valor de um componente RGB.

- Sensor de Profundidade: um emissor de luz quase infra-vermelha espalha um padrão pseudo-aleatório sobre o ambiente e o sensor compara a luz refletida com um padrão gravado no firmware. Dessa maneira, é calculado um mapa de profundidades com resolução de 640 por 480, a uma velocidade de também 30 quadros por segundo. Cada valor tem tamanho 16 bits e não representa uma distância de forma linear. Um processo chamado Registration alinha os pixels da câmera com o do sensor.

- Motor: rotaciona o aparelho em alguns graus na vertical para acompanhar um usuário.

- Microfones: são 4 uniformemente distribuídos na parte inferior, que permitem um processamento de som mais avançado, em geral para melhorar sua qualidade, através de, por exemplo, redução de ruído ambiente e cancelamento de eco.

O grande trunfo dele foi conseguir calcular com precisão de centímetros a distância de objetos e pessoas na cena dentro de um intervalo de 60cm a 5m em tempo real, com hardware de baixo custo. Como se usa um conector USB (e uma fonte de energia extra), foi possível adaptá-lo para usar em computadores. Apenas o que se precisava era um driver que funcionasse corretamente.

2.2 OpenNI

OpenNI ou Open Natural Interaction é uma organização criada em Novembro de 2010 com o objetivo de melhorar a compatibilidade e interoperabilidade de dispositivos de interface natural, aplicações e middleware. Um de seus principais membros é Primesense, a companhia israelense responsável pela tecnologia de sensoriamento 3D (Light Coding) usada no hardware do Kinect.

Como um primeiro produto, a organização disponibilizou um framework open source - OpenNI framework, ou simplesmente OpenNI - que provê uma API para escrever aplicações utilizando interação natural. Está disponível para Windows, Mac e Linux e é escrito originalmente em C++. Por outras experiências envolvendo o Kinect, C# e para abranger uma maior quantidade de usuários, decidiu-se por utilizar um wrapper para a plataforma .NET (disponível gratuitamente para Windows), apesar da escassa documentação existente até então.

A API serve como uma camada intermediária entre dispositivos de baixo nível (câmeras, sensores, microfones e etc.) e soluções middleware de alto nível (algoritmos de visão computacional, por exemplo). Os módulos podem ser registrados e selecionados para criar Production Chains, que nada mais são que sequências de processamento que se iniciam com a captura de dados do ambiente e se estendem até o resultado final que será utilizado pela aplicação.

Gravação é um recurso importante, principalmente para realizar testes com objetos mock. O fluxo de dados é salvo em um arquivo do tipo .oni e pode ser lido posteriormente até como se fosse um vídeo.

Para utilizar o Kinect, é utilizado um driver disponibilizado pela própria PrimeSense.

2.3 NITE

O middleware NITE, outro produto desenvolvido pela empresa, interpreta os dados de profundidade do sensor, criando uma matriz com um mapa de distância em milímetros. Com o auxílio de algoritmos de visão computacional

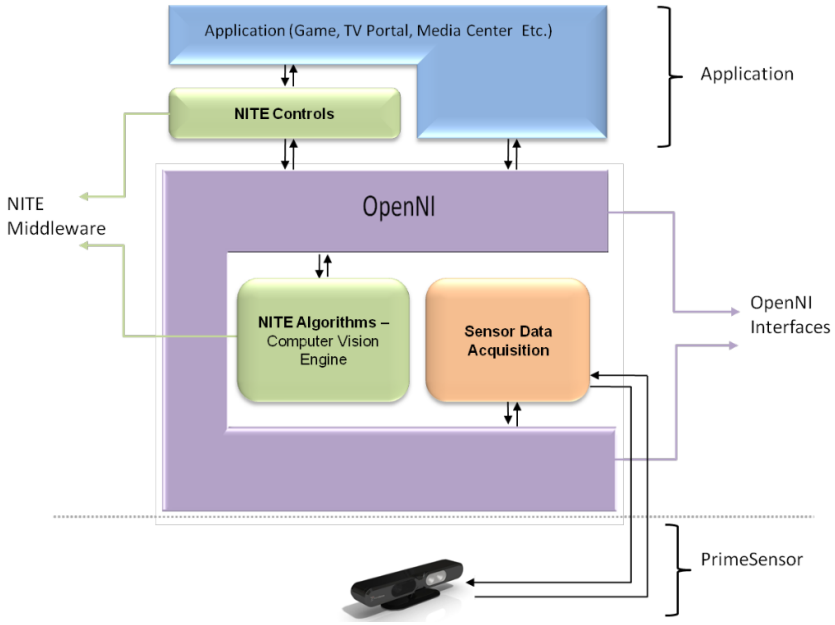

Figura 2: Arquitetura generalizada de programas que utilizam o OpenNI

e processamento de imagens, dispara eventos relacionados ao usuário: quando algum entra ou sai de cena.

Sua principal funcionalidade é o rastreamento de usuários ativos: estimação da posição no espaço 3D de pontos importantes do corpo humano, denominados joints ou articulações. Exemplos são: cabeça, mão, ombro, cotovelo e joelho.

Para que um usuário passivo (só detectado) se torne ativo (sendo rastreado), é preciso passar por um processo de calibragem: manter os braços na posição Psi, que leva esse nome pela semelhança da silhueta de uma pessoa nessa pose com a forma da letra grega Ψ. Depois de alguns segundos, a calibragem estará completa.

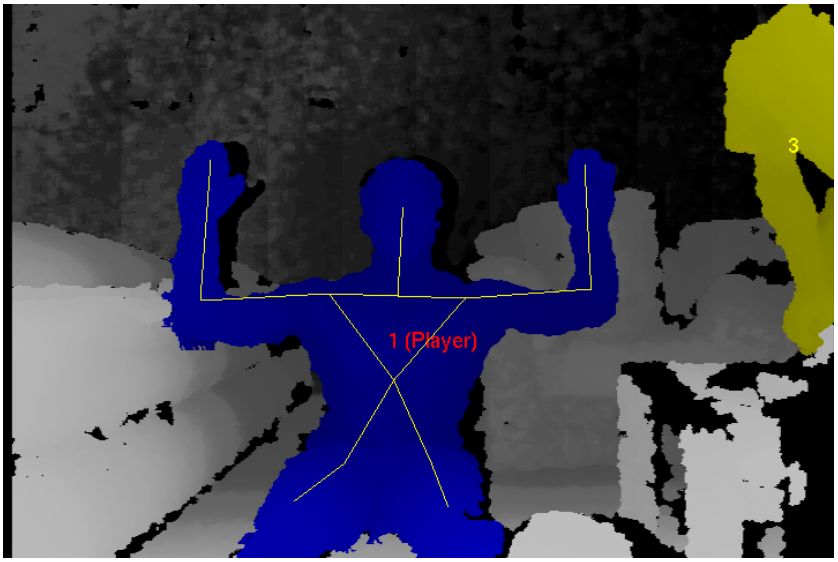

Figura 3: Imagem da pose de calibragem psi gerada com o mapa de profundidades e o de segmentação de usuário

3 Conceitos

3.1 Reconhecimento de Gestos

Reconhecimento de gestos é uma tarefa complexa, que envolve análise e modelagem do movimento, reconhecimento de padrões e aprendizado de máquina. Em alguns casos, até estudos linguísticos. A princípio, eles podem ser formados por qualquer parte do corpo humano, todavia faremos interpretação de apenas uma das mãos, em particular a mão direita. Diversos estudos se baseiam em reconhecimento facial e outras combinações, porém os algoritmos para tais requerem estruturas complexas que fogem ao escopo do bacharelado.

Em trabalhos anteriores realizados na área de interação humano-computador, gestos foram aplicados das mais diversas maneiras:

- tradução de linguagem de sinais
- controle de mundos virtuais
- controle de robôs móveis
- controle musical
- interface de computador alternativa para aqueles que possuem deficiências físicas
- videogames

E obtidos de muitas formas:

- câmera
- luva
- acelerômetro
- giroscópio

O que todos possuem em comum é que, independente do objetivo final e do modo como os dados foram obtidos, é necessário que haja um sistema intermediário capaz de modelar os gestos, o que será realizado com o Modelo Oculto de Markov.

Podemos definir um formalmente gesto como uma sequência temporal $G = (G_1, ..., G_N)$, onde cada G_t é uma pose no instante de tempo t. Uma

pose é $P = \{P_1, ..., P_N\}$, uma coleção de vetores característicos, onde P_i é um vetor característico e i é uma articulação do corpo de um usuário.

No nosso caso, a pose só consiste do vetor característico relaciona à mão direita.

3.2 Extração de características

$V = (V_1, ..., V_N)$ é dito um vetor característico, ou vetor de características, se cada V_i representa uma característica de uma articulação de um usuário num dado instante, isto é, velocidade, posição, aceleração, orientação, etc.

Usamos dois tipo de vetores característicos, ambos tridimensionais: velocidade (vetor formado por duas posições consecutivas) e orientação (velocidade normalizada).

3.3 Filtragem

Os dados obtidos dos dispositivos nem sempre são relevantes porque a alteração pode ser mínima entre um estado e outro. Por isso, usamos dois tipos de filtros.

1. Filtro nível 1: filtra os dados puros fornecidos pelo framework. Se uma nova posição só está a uma distância menor que 5cm da anterior, ela é descartada. Senão, ela passa pelo filtro e a última posição observada é atualizada.

2. Filtro nível 2: é um filtro que compara se a norma da diferença de dois vetores característicos consecutivos é menor que um dado coeficiente de similaridade. Se for, o último vetor extraído é descartado. Senão, ele passa pelo filtro e o último vetor de características observado é atualizado.

3.4 Quantização

A estrutura que apresentaremos a seguir para representar um gesto, precisa que uma pose seja uma variável discreta que assuma valores finitos. Por isso, temos que transformar os vetores característicos em números naturais.

Um quantizador é uma função

$$Q : \mathbb{R}^m \mapsto \mathbb{Z}_n, \text{ em que } n, m \geq 1.$$

Divideremos o espaço R^m em n partições $\{1, ..., n\}$, com cada uma possuindo um representante c_i. Definimos o quantizador como uma função que

determina a partição de um vetor característico v calculando a distância dele a todos os representantes c_i e escolhendo a partição i que tem o representante mais próximo.

$$Q(v) = argmin_i \|v - c_i\|$$

Para atualizar os representantes, o algoritmo de Lloyd (K-Means, K-Médias) é utilizado. Ele consiste de redefinir os representantes como os centróides dos vetores pertencentes à sua partição. A seguir, as partições são recalculadas e os centróides também. A convergência ocorre quando não há mais mudança.

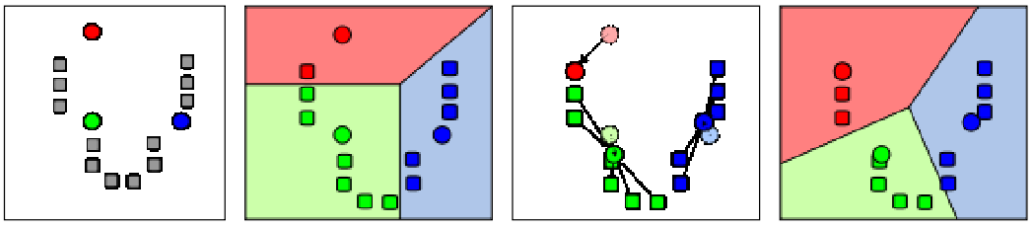

Figura 4: Algoritmo de Lloyd. Círculos são representantes; quadrados serão aglomerados

Para que um particionamento seja eficiente, é crucial a sua inicialização. Uma maneira é atribuir valores aleatórios, com o risco de se obter maus agrupamentos. Outra é pré-definir representantes uniformementes distribuídos pelo espaço, com a restrição de que o número deles deve ser pré-definido.

A segunda alternativa foi implementada. No espaço bidimensional, temos 8 vetores unitários partindo da origem separados por 45 graus, sendo dois deles vetores canônicos. Se imaginarmos que estão contidos num círculo unitário, e queremos expandir os representantes para o espaço tridimensional, podemos intersectar 3 desses círculos ortogonalmente, cada um pertencendo a um plano (XY, YZ, XZ). Desse modo, teremos 18 centróides, e não 24, pois há sobreposições que são descartadas. Para expandir para a quarta dimensão, basta intersectar quatro esferas ortogonalmente, e assim por diante.

Depois de passar pelo quantizador, o gesto é uma sequência de números naturais.

3.5 Modelo Oculto de Markov

Em um modelo de Markov comum, uma variável aleatória assume valores distintos (estados), dependendo apenas do anterior (propriedade de Markov).

O Modelo Oculto de Markov (Hidden Markov Model, também conhecido por HMM), é um processo duplamente estocástico, em que os estados são ocultos. Seu comportamento só pode ser inferido através de símbolos (ou observações), emitidos um para cada valor do estado atual, em função de sua distribuição de probabilidade de emissão.

É utilizado largamente em reconhecimento de padrões, como gestos, escrita, fala e em bioinformática.

Podemos definir formalmente um Modelo Oculto de Markov como uma tupla $\lambda = (N, M, A, B, \pi)$ com os seguintes elementos:

- N é o número de estados ocultos. $S = \{1, ..., N\}$, o espaço de estados. Denotamos por X_t o estado no tempo t.
- M é o número de símbolos observáveis. $V = \{1, ..., M\}$, o espaço das observações. Denotamos por Y_t a observação no tempo t.
- Uma matriz de probabilidades de transição $A = \{a_{ij}\}$, onde $a_{ij} = P(X_{t+1} = j | X_t = i)$, com $1 \leq i, j, \leq N$. A matriz satisfaz as restrições estocásticas: $a_{ij} \geq 0$ e $\Sigma_{j=1}^{N} a_{ij} = 1$.
- Uma matriz de probabilidades de emissão $B = \{b_j(k)\}$, onde $b_j(k) = P(Y_t = k | X_t = j)$, com $1 \leq j \leq N$ e $1 \leq k \leq M$. A matriz satisfaz as restrições estocásticas: $b_j(k) \geq 0$ e $\Sigma_{k=1}^{M} b_j(k) = 1$.
- A distribuição inicial $\pi = \{\pi_i\}$, onde $\pi_i = P(X_1 = i)$, com $1 \leq i \leq N$.

Nota-se que A e π formam uma Cadeia de Markov Oculta e que A e B formam um processo aleatório cada.

Uma forma de representação implícita de λ é (A, B, π), já que N e M podem ser inferidos a partir dos parâmetros A e B.

A topologia define a estrutura interna dos estados do modelo. Existem dois tipos principais.

A ergódica, em que a distribuição inicial tem probabilidade não nula para todos os estados, que são alcançados por todos em apenas um passo.

$$\pi_i > 0 \text{ e } a_{ij} > 0$$

E a de Bakis, em que só existe um estado inicial e cada estado só pode ter probabilidade de transição não nula para os Δ estados seguintes e ele mesmo.

$$\pi_1 = 1,\ \pi_i = 0,\ 1 < i \leq N$$
$$a_{ij} = 0,\ j < i \text{ ou } j > i + \Delta$$

Em particular, quando $\Delta = 1$, temos o modelo Left-Right.

Outros tipos especiais de HMMs descritos na literatura incluem Modelos Ocultos de Markov Acoplados (CHMM) e modelos com distribuição de probabilidade de emissão contínua (em geral gaussiana).

Uma situação que exemplifica bem uma modelagem por HMM é a de João e Maria. Eles são dois amigos que moram em lugares muito distantes e se comunicam pela Internet. Maria conta todo dia (t) a João o que ela fez (Y_t), se andou, leu algum livro ou foi às compras. E ele tenta adivinhar como estava o tempo no dia (X_t), chuvoso ou ensolarado, pois sabe quais são as probabilidades de ela realizar cada atividade dado o tempo (B), conhece também a previsão do tempo (A), mas não sabe o tempo atual na cidade dela.

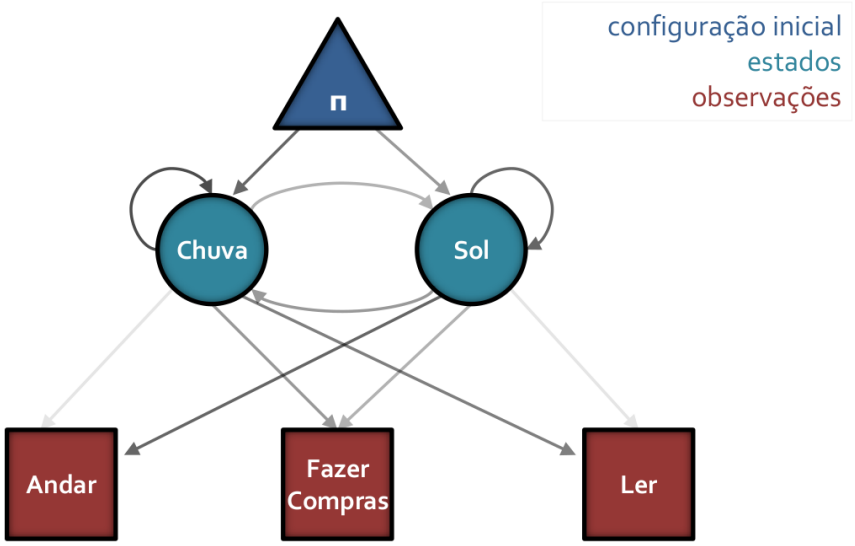

Figura 5: Exemplo de João e Maria. Quanto maior a opacidade das setas, mais próxima de 1 é a probabilidade.

Este problema é conhecido como o problema da decodificação, e pode ser resolvido eficientemente com o algoritmo de Viterbi. No entanto, dois outros problemas merecem maior detalhamento por se mostrarem mais relevantes. Estimação, que leva ao reconhecimento de gestos, e aprendizagem, que leva ao treinamento.

No contexto deste trabalho, os estados ocultos modelam os gestos em si e as observações o que é possível extrair deles como informação.

3.5.1 Estimação

Descrição: Computar eficientemente a probabilidade de uma sequência de observações dado um HMM.

$$P(O|\lambda) = P(O_1, ..., O_T|\lambda)$$

Note o "eficientemente", já que só computar esse valor é trivial porém com complexidade exponencial. Uma alternativa é utilizar o conceito de programação dinâmica e definir uma variável denominada forward.

$$\alpha_t(i) = P(O_1, ..., O_t, X_t = i|\lambda)$$

Ela indica a probabilidade de λ emitir os t primeiros elementos da sequência observada O e estar no estado i no instante t. Podemos reescrêve-la usando indução.

$$\alpha_1(i) = \pi_i b_i(O_1), 1 \leq i \leq N$$
$$\alpha_{t+1}(j) = [\Sigma_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(O_{t+1}), 1 \leq t \leq T-1, 1 \leq j \leq N$$

Na base, o instante é 1. Então a probabilidade de emitir o símbolo O_1 é a de começar num estado qualquer e emiti-lo. No passo, a probabilidade de emitir uma sequência de tamanho $t+1$ e parar no estado j é o mesmo que observar a subsequência de tamanho t e parar num estado qualquer i, transitar de i para j e a partir dele emitir o símbolo restante O_{t+1}.

Então é claro que a probabilidade de observar toda a sequência é a somatória das probabilidades de observar toda a sequência terminando em qualquer estado.

$$P(O|\lambda) = \Sigma_{i=1}^{N} \alpha_T(i)$$

Faremos o mesmo com a variável backward, que indica a probabilidade dos últimos $T-t$ símbolos serem emitidos dado que λ está no estado i no tempo t.

$$\beta_t(i) = P(O_{t+1}, ..., O_T|X_t = i, \lambda)$$

De forma indutiva, temos.

$$\beta_T(i) = 1, 1 \leq i \leq N$$
$$\beta_t(i) = \Sigma_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), 1 \leq t \leq T-1, 1 \leq j \leq N$$

Convencionamos a base como 1 e definimos o passo de modo análogo ao forward. A fórmula da estimação pode ser reelaborada alternativamente como:

$$P(O|\lambda) = \Sigma_{i=1}^{N} \pi_i \beta_1(i) b_i(O_1)$$

É fácil ver que em ambos os casos a complexidade do algoritmo é $O(N^2T)$.

3.5.2 Aprendizagem

Descrição: Determinar um novo modelo a partir do existente que maximize (localmente) a probabilidade de se obter uma sequência observada.

$$P(O|\overline{\lambda}) > P(O|\lambda)$$

Por que localmente? Porque não se conhece nenhum algoritmo eficiente para o caso global. O algoritmo descrito a seguir é conhecido como Baum-Welch. Para apresentá-lo, definiremos mais duas novas variáveis.

$$\gamma_t(i) = P(X_t = i|O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$$

$$\xi_t(i,j) = P(X_t = i, X_{t+1} = j|O, \lambda) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

O que elas significam? Se fizermos uma somatória com elas fica mais fácil de entender.

$$\Sigma_{t=1}^{T-1}\gamma_t(i)$$

Esse é o número esperado de visitas ao estado i ou transições a partir do estado i.

$$\Sigma_{t=1}^{T-1}\xi_t(i,j)$$

Esse é o número esperado de transições a partir do estado i para o estado j. Reestimamos, então, os parâmetros do HMM λ, criando um novo modelo $\overline{\lambda} = (\overline{\pi}, \overline{A}, \overline{B})$. A probabilidade de transição de i a j será proporcional à esperança de visitas ao estado j dado que passamos por i. O mesmo ocorre com as emissões.

$$\begin{aligned}
\overline{\pi_i} &= \gamma_1(i) \\
\overline{a_{ij}} &= \frac{\Sigma_{t=1}^{T-1}\xi_t(i,j)}{\Sigma_{t=1}^{T-1}\gamma_t(i)} \\
\overline{b_j(k)} &= \frac{\Sigma_{t=1,O_t=k}^{T}\gamma_t(j)}{\Sigma_{t=1}^{T}\gamma_t(j)}
\end{aligned}$$

É fácil observar que a complexidade do algoritmo é $O(N^2T)$. Mas ele é executado várias vezes até o momento em que o modelo não consegue mais ser melhorado substancialmente.

3.6 Classificador

Suponha que temos um conjunto de n modelos já treinados $\{\lambda_1, ..., \lambda_n\}$ e queremos classificar um gesto O como pertencente a algum λ_i, isto é, queremos escolher o modelo λ^* dentre os existentes que tenha a maior probabilidade *a posteriori* de representar O:

$$\lambda^* = argmax_{\lambda_i} P(\lambda_i|O)$$

No entanto, para cada λ_i, apenas conhecemos a probabilidade calculada pelo problema da estimação:

$$P(O|\lambda_i)$$

Podemos usar o Teorema de Bayes para reescrever λ^*:

$$\lambda^* = argmax_{\lambda_i} \frac{P(\lambda_i)P(O|\lambda_i)}{P(O)}$$

Mas supondo que as probabilidades *a priori* $P(\lambda_i)$ são equivalentes e sabendo que $P(O)$ sempre possui o mesmo valor para todo λ_i, o classificador pode ser redefinido como:

$$\lambda^* = argmax_{\lambda_i} P(O|\lambda_i)$$

Dessa maneira, para reconhecer um gesto basta resolver o problema da estimação para cada modelo e escolher o de maior probabilidade.

4 Desenvolvimento

A ideia inicial do projeto era exatamente fazer uma aplicação com o Kinect que reconhecesse gestos. Porém, não se sabia de que maneira isso seria feito.

Depois de compreender o funcionamento do hardware e pesquisar os diversos programas desenvolvidos, chegou-se a uma conclusão: a maioria deles consistia de manipulação direta de objetos virtuais. Essa classe de gestos é conhecida como *online*, já que eram interpretados enquanto eram executados. A minoria que realmente se baseava em gestos *offline*, era *hardcoded*. Queríamos também uma técnica que possibilitasse ao sistema aprender com qualquer usuário, sem a necessidade de um programador.

Iniciamos uma busca por referências desse problema na literatura, com qualquer que fosse o dispositivo (câmera, luva, acelerômetro). A resposta era utilizar um modelo probabilístico para representar um gesto. Podendo ser uma Rede Neural ou Modelo Oculto de Markov, entre outros.

Nos apronfudamos no estudo de Markov só para descobrir que ele era a ponta do *iceberg*. Existia toda uma sequência de processamento que deveria ser implementada para chegar nele.

Invertemos a ordem de estudo. O primeiro passo foi descobrir como ligar o Kinect no computador. Parecia uma tarefa simples, mas não era só conectar o dispositivo a uma porta USB. Precisava de um driver, que não existia oficialmente. A comunidade de software livre, por meio de engenharia reversa, desenvolveu um que não funcionava tão bem. O bom é que a mesma empresa que criou a tecnologia acabou disponibilizando um para Windows.

Um fruto importante das pesquisas com os *hacks* foi tomar conhecimento das ferramentas mais utilizadas. Entre diversos candidatos, o OpenNI se mostrou o mais atrativo, pois era o mais utilizado e com os recursos necessários para o trabalho.

A produção do projeto se deu por iterações, seguindo a metodologia ágil de desenvolvimento. A cada semana eram determinadas as tarefas mais importantes a serem realizadas, sempre com um protótipo funcional pronto ao fim de cada uma delas.

A primeira tarefa foi implementar uma janela simples que mostrasse as imagens do dispositivo. A interface de usuário foi desenvolvida com o subsistema gráfico WPF da plataforma .NET framework. O software foi escrito na linguagem de programação C#. Aos poucos, adicionamos outros recursos sobre as imagens: número de identificação, estado de rastreamento e destacamento de esqueletos das pessoas.

Para simplificar o reconhecimento de gestos, determinou-se que apenas um dos usuários ativos seria observado pelo sistema. Para tanto, devíamos es-

colher qual deles merecia o foco de atenção. Existem vários critérios possíveis: o mais próximo do aparelho, o mais distante, o que apareceu primeiro, o que apareceu por último e o que estiver dentro de uma região pré-determinada. A última opção acabou sendo descartada porque seria difícil para o usuário comum parametrizar regiões.

A funcionalidade de espelhar as imagens obtidas do ambiente foi desabilitada na versão final. No início, ela estava presente somente no arquivo de configuração que era lido pelo programa quando ele começava a ser carregado. Mas o que acontecia na verdade era um espelhamento na geração dos dados. Então, a configuração foi excluída e uma propriedade dos nós geradores de dados do OpenNI - Mirror - passou a ser utilizada. Desse modo, era possível alterar durante a execução. Só que outro problema surgiu: usuários ativos eram perdidos durante a troca. Optou-se por remover a funcionalidade e deixar a imagem sempre espelhada, já que é o jeito mais normal de uma pessoa se ver.

Como já foi explanado, um requisito de rastreamento é a calibragem. Querendo deixar o processo mais simples para o usuário, gravamos as medidas de um esqueleto padrão num arquivo e o atribuíamos automaticamente a cada evento "novo usuário"disparado. No entanto, aconteceu algo semelhante com o espelhamento: no momento de alternância entre os dois modos, ocorria *memory leak* por algum motivo não descoberto. Decidimos manter só a calibragem padrão, até porque ela garante mais precisão.

Então, chegamos à parte importante. O problema principal agora era definir quando um gesto começa e quando ele termina.

Um gesto tem 3 fases: preparação, execução e retração. A execução é a parte dinâmica que contém a informação que estamos interessados em extrair. Podemos fazer uma identificação das outras fases, portanto, para segmentar uma sequência de gestos. Para esta tarefa, é possível criar um filtro que detecta quando um movimento saiu do repouso (preparação), ou chegou nele (retração). Porém, se um movimento conter um trecho de repouso, o filtro pode levar a erros ou ambiguidades.

Outra alternativa é determinar explicitamente o início e o fim, com algum tipo de evento externo ao gesto, como um *timer* ou clique do *mouse*. Para simplificar a solução do problema, a segunda abordagem foi adotada.

Uma das funcionalidades desejadas era poder traduzir um gesto para um comando. Contudo, apesar de toda simulação de teclado e mouse ter sido concluída, não tinha como usá-la justamente porque os gestos tinham que ser explicitamente segmentados.

O próximo passo foi implementar o extrator. Pelos materiais estudados, estava mais evidente que a orientação era uma característica que descrevia bem um movimento, e o OpenNI a fornecia. Na verdade, o valor não era o

que esperávamos. Ele definiu orientação como uma matriz de transformação linear que transformava as coordenadas de um ponto no espaço real centrado no Kinect para um sistema de coordenada centrado na articulação. Logo, tivemos que definir orientação e velocidade como já explicamos nas seções anteriores.

Outra coisa que não funcionou como esperado foi o recurso de gravação. Os dados gerados pelo dispositivo eram gravados com sucesso, mas não era possível fazer a leitura sem interromper o fluxo atual. Tivemos que implementar nossa própria gravação: um simples arquivo de texto com as posições da mão direita do usuário principal.

A quantização com inicialização aleatória era muito ruim. Acabou sendo descartada e o algoritmo de Lloyd passou a ser inicializado conscientemente, com vetores quase que uniformemente distribuídos no espaço das características.

O algoritmo de Baum-Welch aplica o aprendizado de um modelo para apenas uma sequência de observações. Ele teve que ser adaptado para calcular as médias das probabilidades de múltiplas observações e tentar maximizá-las.

O ideal seria que o treinamento fosse incremental. Mas isso significa que todo o conjunto teria que ser guardado e aumentado a cada nova requisição de treino. A complexidade do algoritmo ficaria cada vez maior e afetaria o desempenho. Resolvemos fazer o treinamento apenas uma vez e com um número fixo de amostras requeridas.

Os algoritmos estavam implementados corretamente, segundo a teoria, mas não funcionavam na prática porque acontecia underflow. Um produto de muitos termos pequenos (entre 0 e 1) resultava em 0, porque em um momento o valor ficava menor que o menor número representável.

Várias soluções em conjunto foram utilizadas para minimizar esta limitação da representação em ponto flutuante. Uma delas foi usar logaritmo para transformar o produto em soma e distribuir melhor os valores, que se antes estavam contidos no intervalo $[0, 1]$, agora eram menores que 0 e numa escala maior.

A adaptação mais importante foi reescrever os algoritmos com coeficientes de escalamento, para que os valores não diminuíssem tanto a cada operação. Com essas mudanças, a aplicação começou a funcionar corretamente. Finalmente, a testamos para medir sua eficácia.

5 Resultados

Os gestos testados foram quatro: círculo, arco, hélice e quadrado. Quatro participantes realizaram quarenta vezes cada um deles e à cada execução foi atribuída um número inteiro, começando de um. Os gestos ímpares formaram o conjunto de treinamento e os pares o de reconhecimento.

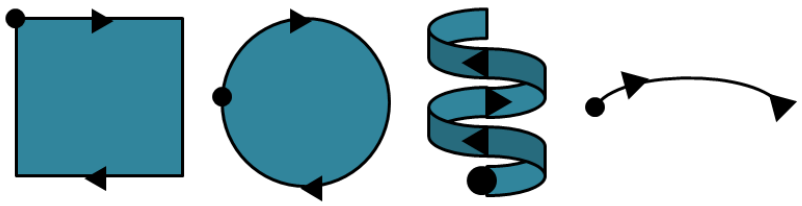

Figura 6: Gestos do ponto de vista do usuário. O ponto indica o início e a seta o sentido. Da esquerda para a direita: quadrado, círculo, hélice e arco (vai e volta).

O objetivo do primeiro conjunto de testes foi determinar a estrutura mais apropriada do modelo oculto de Markov, isto é, o número de estados ocultos e a topologia que produzisse alta taxa de certo com a menor quantidade de treinamento possível.

A escolha do extrator e do coeficiente de similaridade do filtro de nível 2 não teriam muita influência no resultado final, então foram estabelecidos valores padrão: orientação e $0,3$, respectivamente.

A seguir são apresentados os gráficos de linha com o número de acertos para cada subconjunto de treinamento (de ordens 10, 20, 40 e 80) de cada um dos nove modelos.

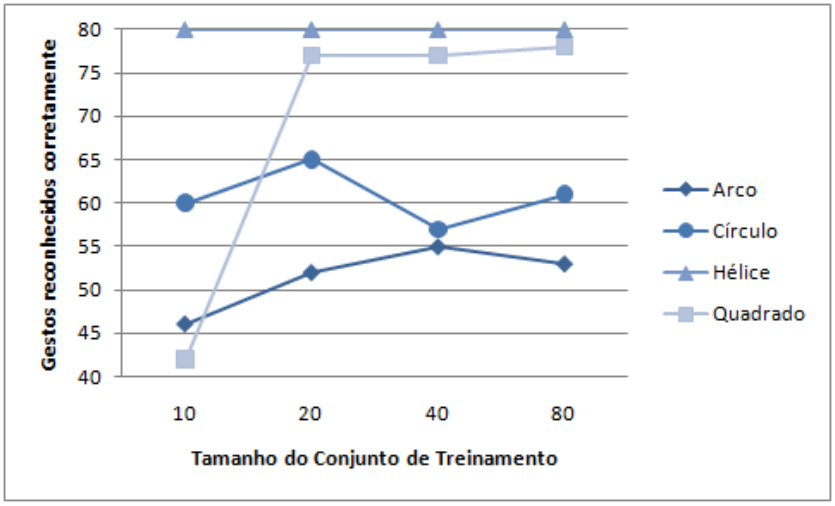

Figura 7: Modelo 1 (topologia ergódica com 5 estados)

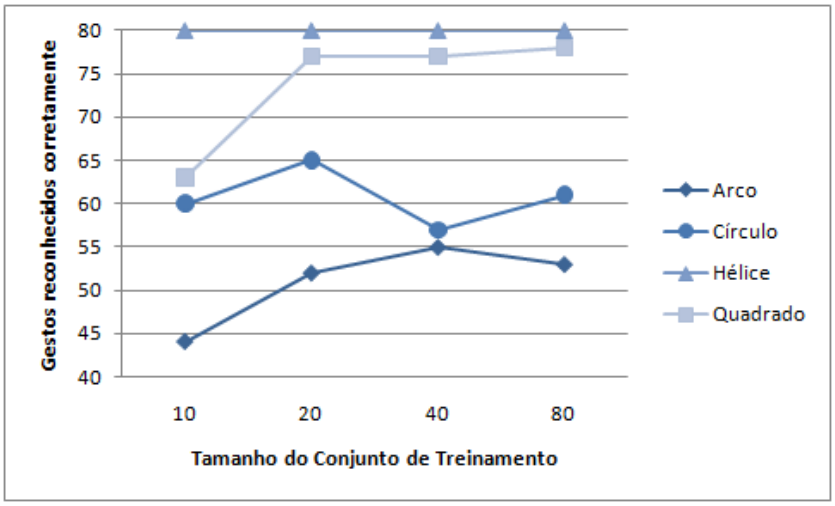

Figura 8: Modelo 2 (topologia ergódica com 10 estados)

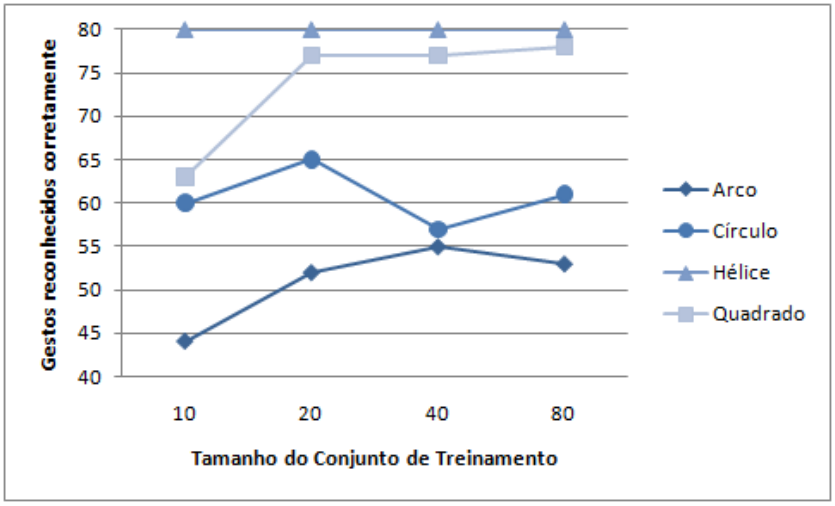

Figura 9: Modelo 3 (topologia ergódica com 15 estados)

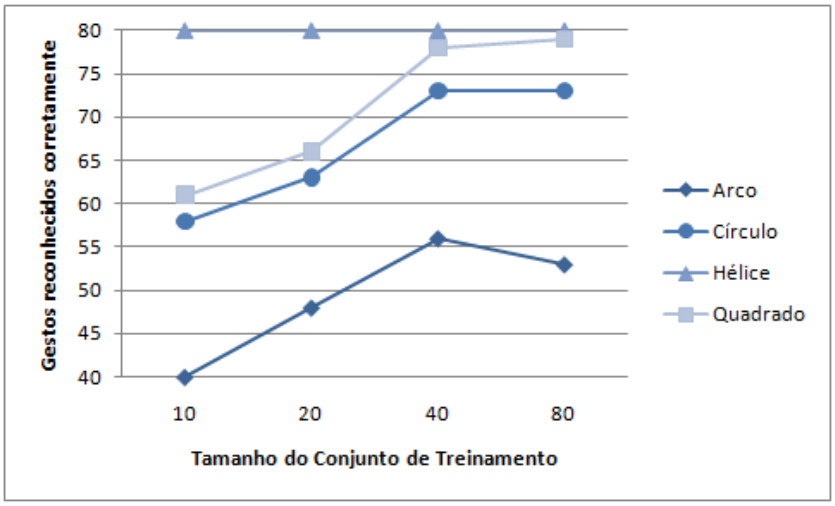

Figura 10: Modelo 4 (topologia de Bakis com delta 1 e 5 estados)

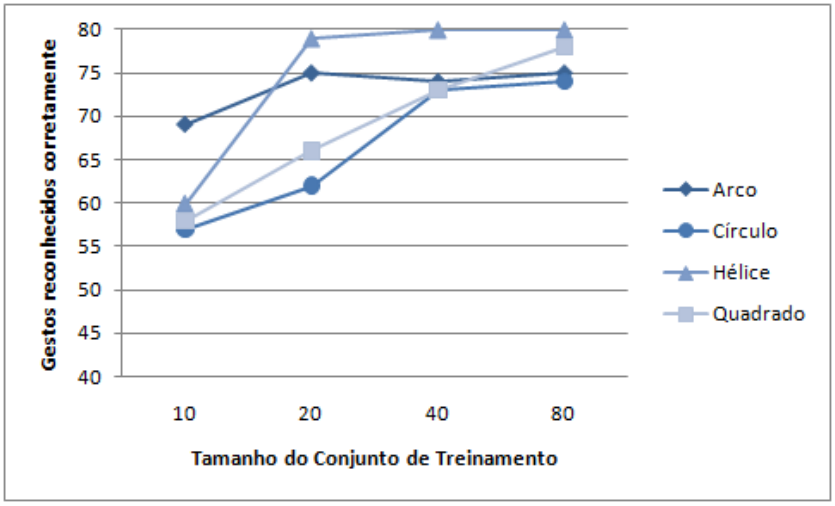

Figura 11: Modelo 5 (topologia de Bakis com delta 1 e 10 estados)

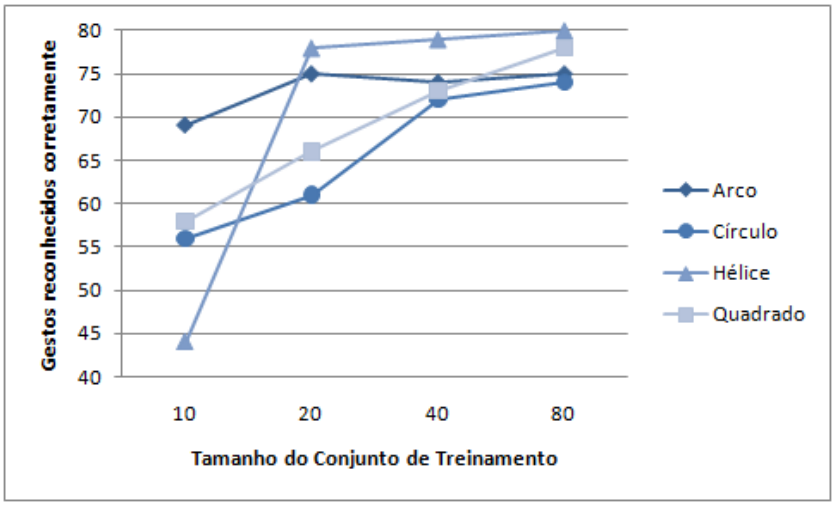

Figura 12: Modelo 6 (topologia de Bakis com delta 1 e 15 estados)

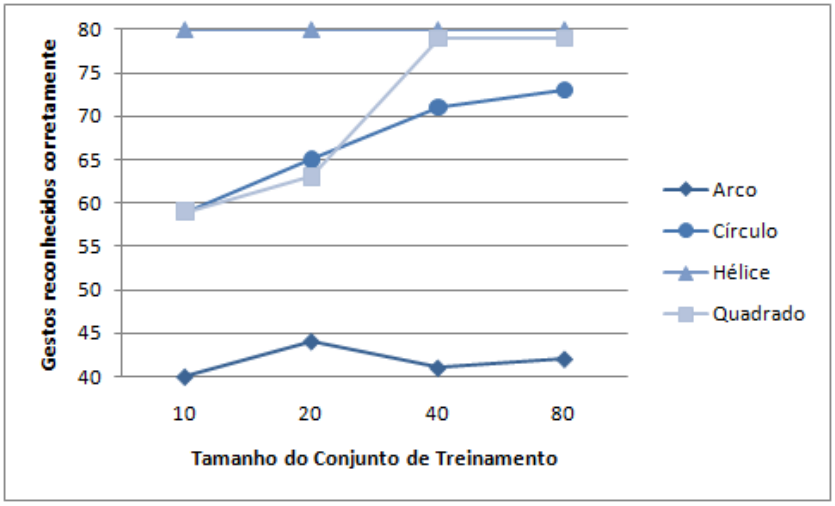

Figura 13: Modelo 7 (topologia de Bakis com delta 2 e 5 estados)

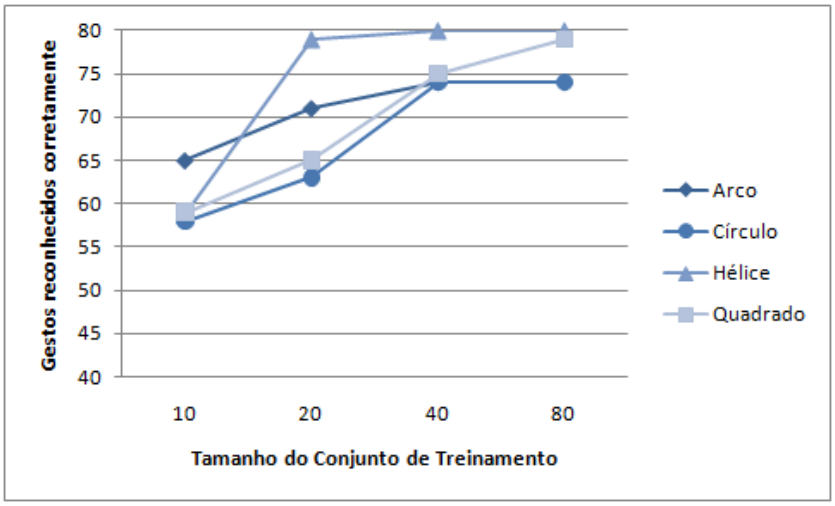

Figura 14: Modelo 8 (topologia de Bakis com delta 2 e 10 estados)

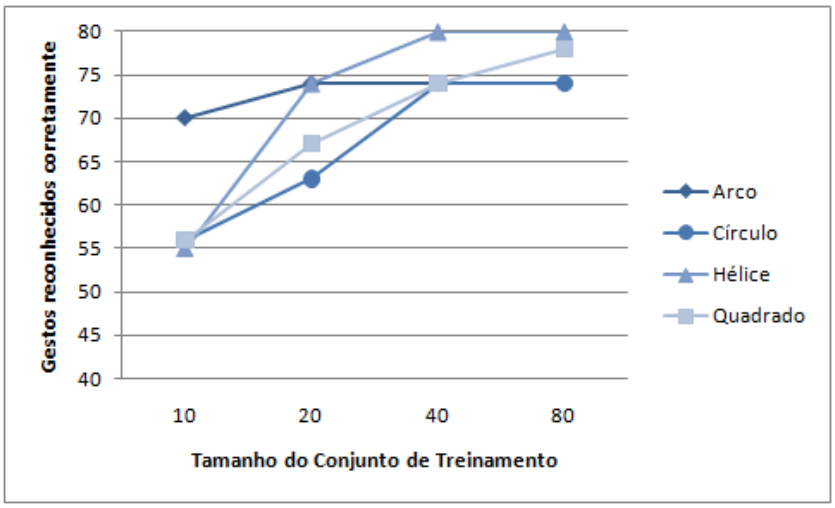

Figura 15: Modelo 9 (topologia de Bakis com delta 2 e 15 estados)

Observa-se que, independentemente do modelo, hélice e quadrado são os gestos mais fáceis de serem reconhecidos. E que o arco é o mais difícil, o que não deixa de ser surpreendente, pois é o mais simples de ser executado.

Outro ponto importante a se notar é que uma melhora na probabilidade devido ao treinamento não indica necessariamente que a acurácia irá aumentar, já que os outros HMMs também mudam, e nesse caso pode haver uma classificação errônea.

Para melhor comparar, visualizemos cada classe (mesma topologia) de modelo usando como medida a média de acertos.

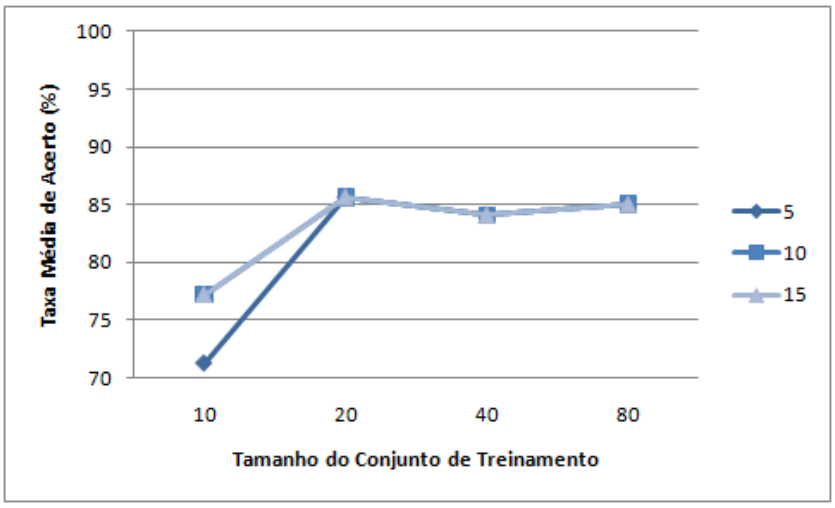

Figura 16: Classe 1 (topologia ergódica)

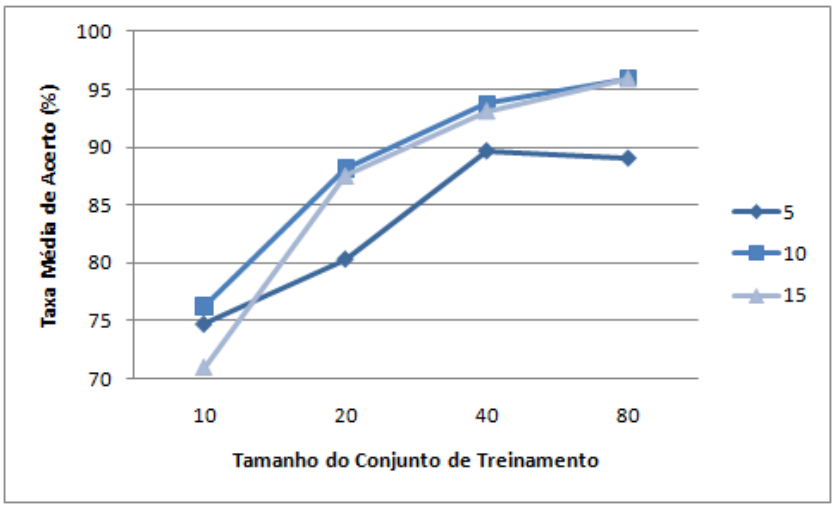

Figura 17: Classe 2 (topologia de Bakis com delta 1)

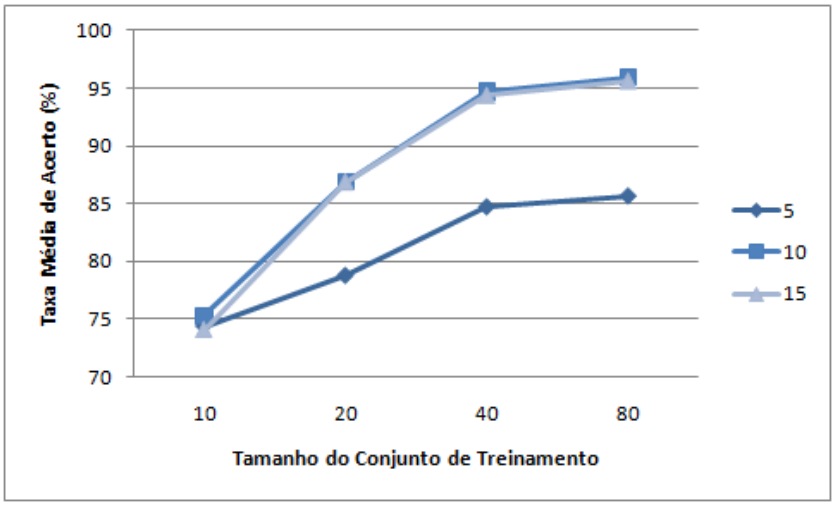

Figura 18: Classe 3 (topologia de Bakis com delta 2)

Na topologia ergódica, a quantidade de estados não fez muita diferença. Uma interpretação possível é que o processo estocástico pode saltar infinitamente com o passar do tempo, enquanto que no outro há um estado final, o que limitaria o número de saltos pelo tamanho da cadeia oculta. Por outro lado, não há indícios de que quanto maior esse limite melhor. Nos testes, 10 e 15 são praticamente iguais para grandes treinos.

A última etapa para escolher o modelo "ideal"é tomar os representantes mais acurados de cada classe.

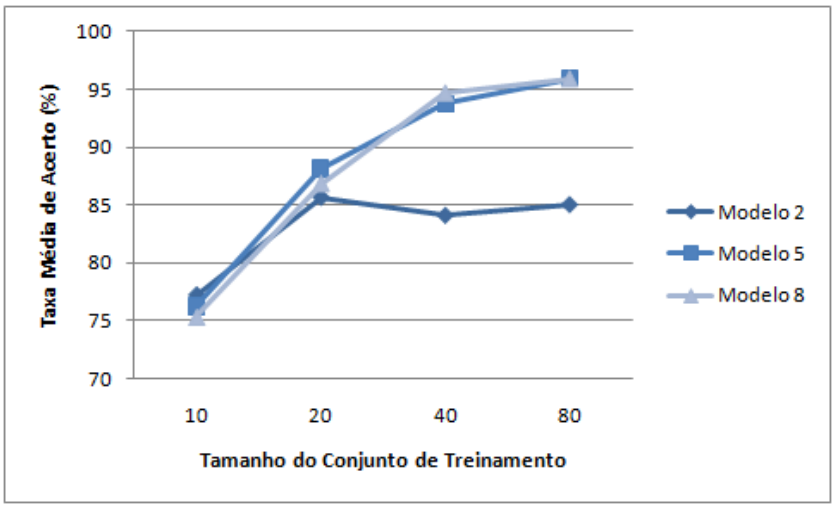

Figura 19: Comparação entre representantes de cada classe

A disputa ficou entre os modelos 5 e 8, fato que condiz com outros trabalhos que afirmam que Bakis modela com mais eficácia propriedades de progressão temporal. As taxas de acerto de ambos foram idênticas para o maior treino. Mas nos dois primeiros (10 e 20), o modelo 5 foi melhor e por isso foi escolhido como o modelo "ideal"a ser usado na versão final do programa.

A seguir, o objetivo foi fazer um ajuste fino nos outros componentes do *pipeline*. Testamos, logo, as combinações dos extratores de orientação (versor gerado por duas posições consecutivas) e velocidade (vetor gerado por duas posições consecutivas) com diferentes valores do coeficiente de similiaridade do filtro de nível 2, 0,25 e 0,5.

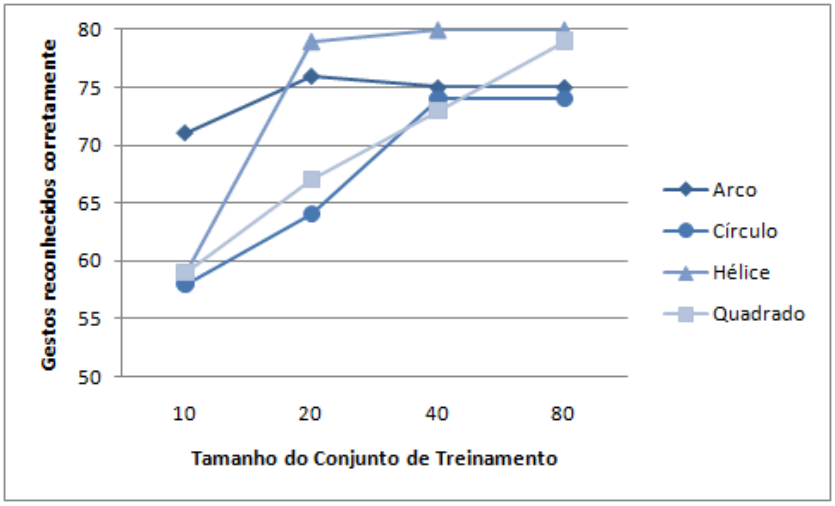

Figura 20: Combinação 1 (Extrator de orientação e similaridade 0,25)

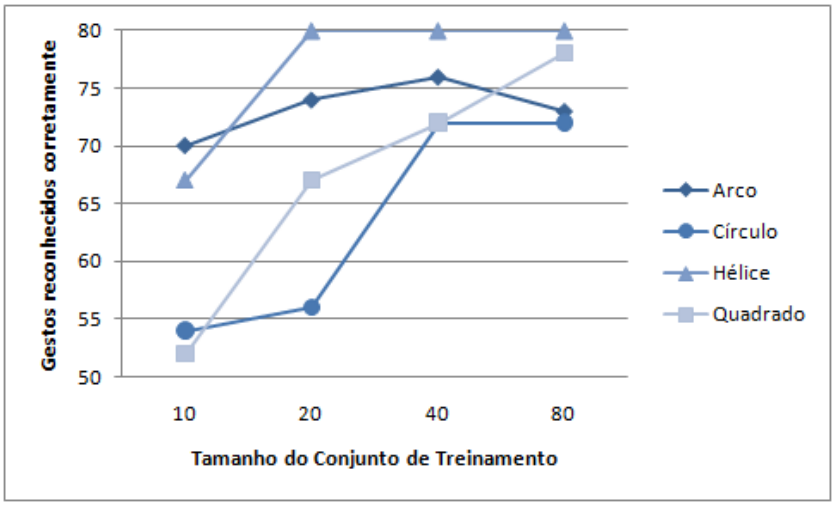

Figura 21: Combinação 2 (Extrator de orientação e similaridade 0,5)

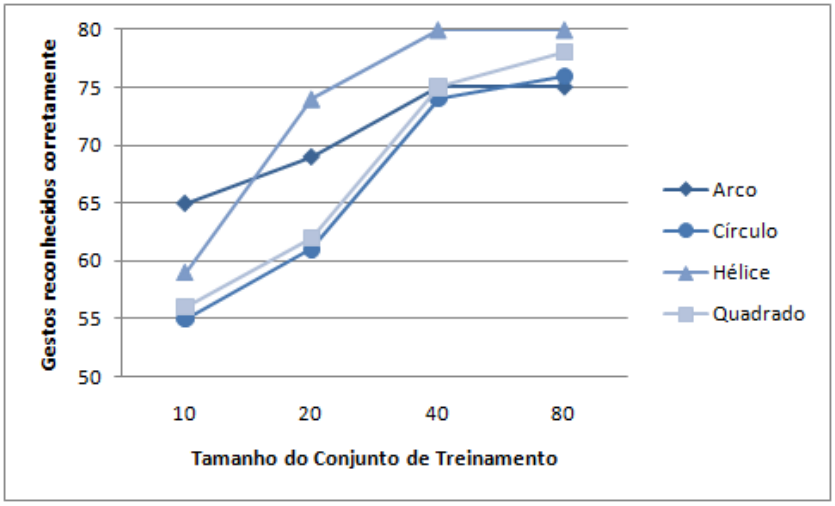

Figura 22: Combinação 3 (Extrator de velocidade e similaridade 0,25)

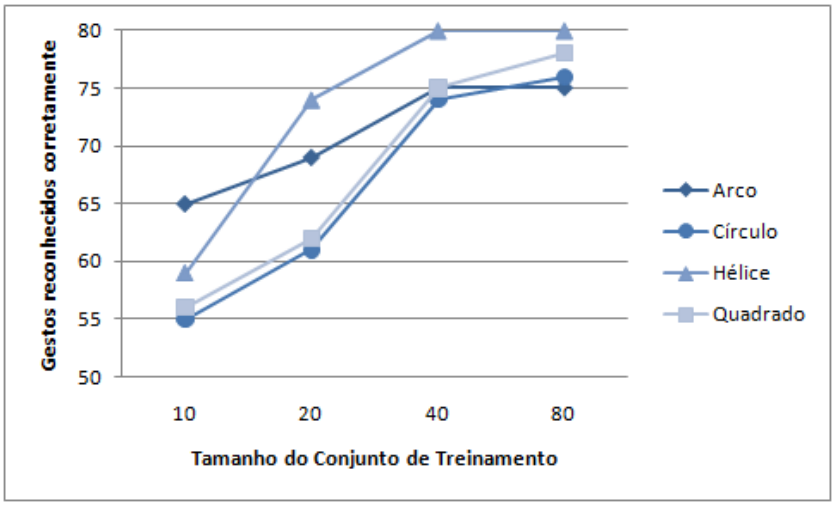

Figura 23: Combinação 4 (Extrator de velocidade e similaridade 0,5)

As combinações 3 e 4 obtiveram exatamente os mesmos resultados. Outro dado interessante é que um círculo é descrito melhor por vetores de velocidade que de orientação.

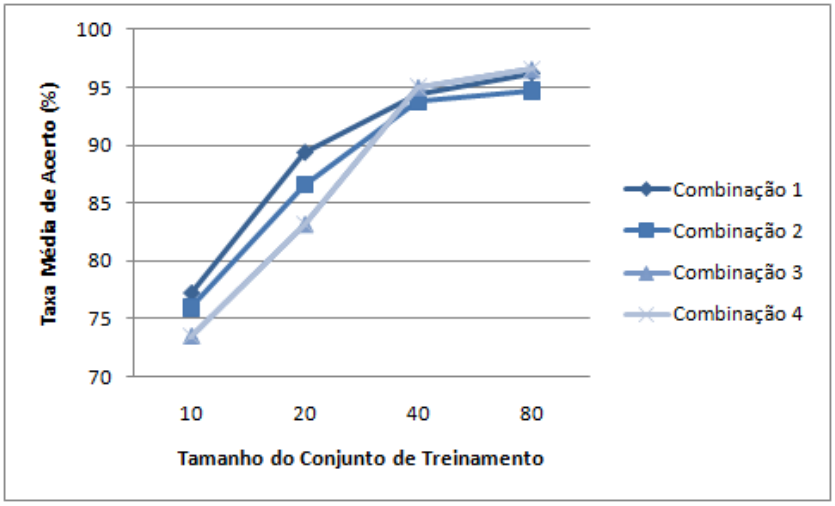

Figura 24: Comparação das combinações

A maior proporção de acertos no reconhecimento foi o da combinação 1, exceto quando o conjunto de treinamento tinha ordem 80. Entretanto pode-se considerar esta diferença desprezível. Por conta disso, as parametrizações dela se tornaram a padrão. Então geramos ainda mais estatísticas: os acertos do sistema para cada usuário em função do treinamento.

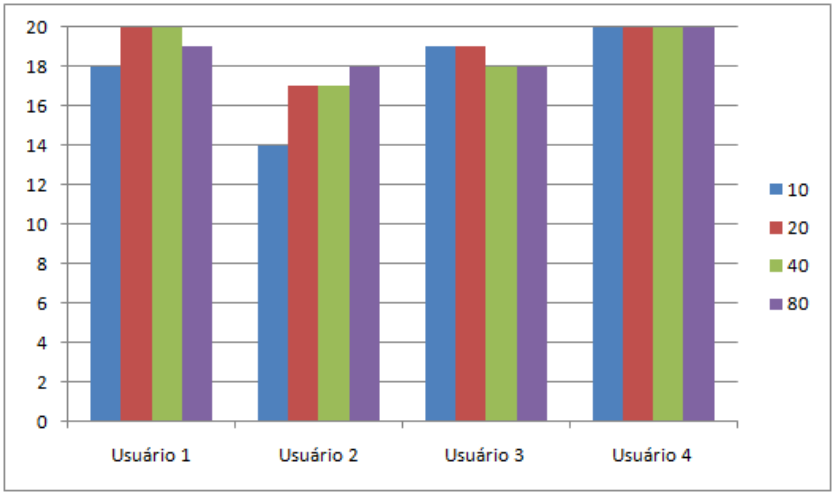

Figura 25: Número absoluto de acertos no reconhecimento do gesto Arco

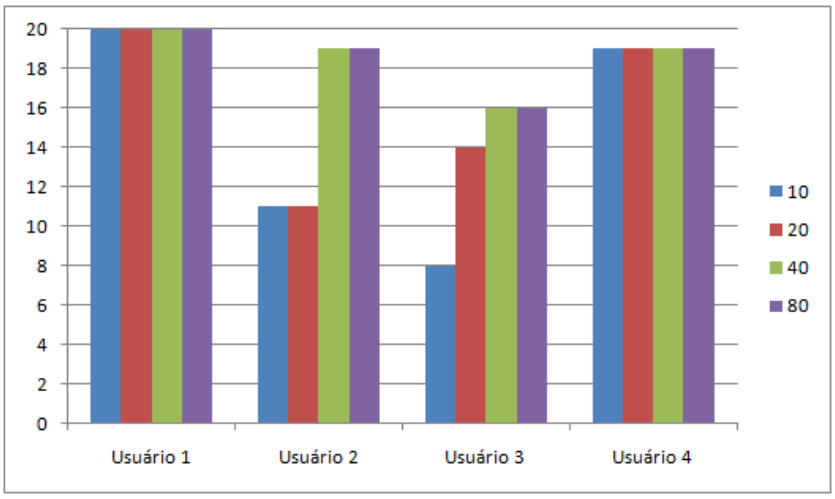

Figura 26: Número absoluto de acertos no reconhecimento do gesto Círculo

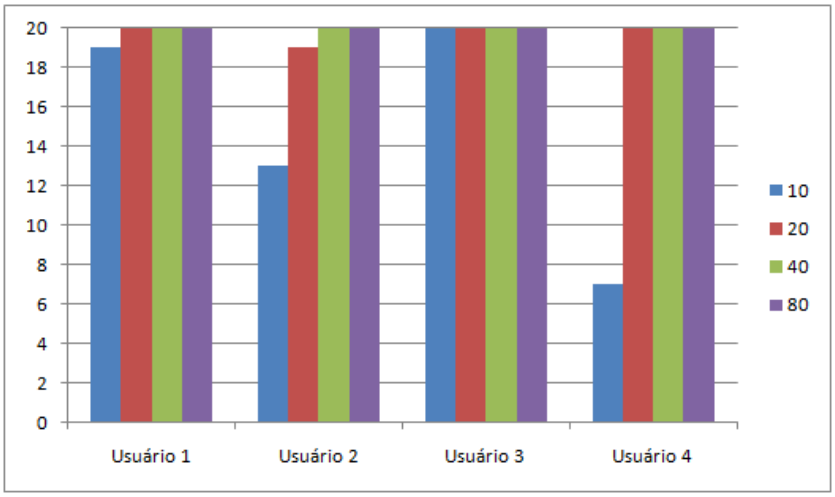

Figura 27: Número absoluto de acertos no reconhecimento do gesto Hélice

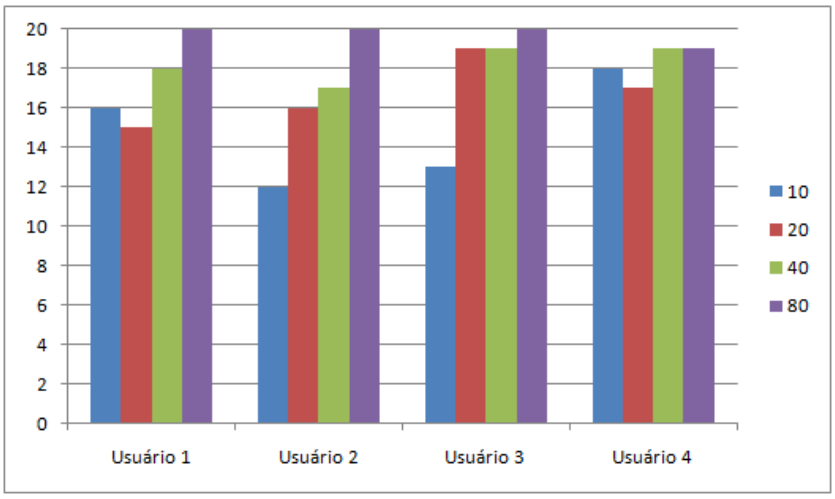

Figura 28: Número absoluto de acertos no reconhecimento do gesto Quadrado

Os gestos mais problemáticos, ou seja, com menor porcentagem de respostas corretas, tiveram um desempenho pior por motivos diferentes. O arco parece ter sido realizado de maneiras notavelmente distintas por todos, enquanto que o círculo mais especificamente pelo usuário 3.

6 Conclusão

Os algoritmos ficaram eficientes o bastante, rodando em tempo-real em máquinas com poder de processamento comum atualmente. De modo geral, os objetivos estipulados inicialmente foram alcançados.

Existem muitos mais parâmetros que ainda podem ser testados em toda a sequência de processamento. Só foram realizados os supracitados porque apresentaram uma porcentagem de acerto bastante satisfatória, com média no melhor caso de aproximadamante 96%, superando, inclusive, as expectativas com este trabalho.

6.1 Possíveis Mudanças

Algumas mudanças mais imediatas seriam:

- A substituição do framework OpenNI pelo Microsoft Kinect SDK, que em sua versão mais recente não exige calibragem dos usuários e possui uma estimação das posições das articulações mais correta.
- Fornecer ao usuário a opção de realizar o treino incrementalmente.
- Exportar gravações serializando uma classe para um arquivo e não em modo texto, o que evita ter que escrever um parser e garante a segurança na leitura. Da mesma forma que é implementada atualmente a exportação/importação de gestos.

Também existem outras mais complexas:

- Utilizar um Modelo Oculto de Markov em que a emissão de cada estado obedece uma função de distribuição de probabilidade contínua.
- Informar que um gesto não foi reconhecido ao invés de responder errado quando há uma probabilidade irrisória de acerto. Isso só acontece em casos extremos, como por exemplo, quando os modelos ainda não estão treinados.
- Sem dúvida, o mais importante é reconhecer gestos continuamente, sem a necessidade de explicitamente determinar o início e o fim de um gesto. É uma tarefa trabalhosa, mas essencial para levar o programa do estágio em que ele se encontra para uma ferramenta de interação homem-máquina poderosa.

Parte II
Subjetiva

1 Desafios e Frustrações

No início, eu não fazia ideia de que o problema de reconhecimento de gestos fosse tão complicado. Estava meio perdido: um leigo no assunto que não sabia por onde começar. Então, o professor Flávio indicou dois trabalhos para leitura e direcionou os meus estudos para Modelos de Markov e Redes Neurais. Acabei optando pelo primeiro, pois aquele nome já me era familiar. Comecei a pesquisar sobre HMMs e só o que apareciam eram teses de mestrado e doutorado com taxas de acerto não muito altas, e apenas com reconhecimento de gestos 2D.

Parecia que quanto mais eu estudava, mais ficava difícil. Chegou um momento que senti que realmente não ia dar para seguir por esse caminho. Mas acabei não desistindo e depois de alguns meses e perto da entrega final, consegui chegar num resultado.

Parte da dificuldade atribuo a implementar todos os algoritmos sem usar nenhuma biblioteca pronta, excetuando, claro, o framework que realizava o processamento das imagens. Outra parte à tarefa de ter que encontrar soluções para problemas que não estavam muito bem compreendidos ou definidos.

2 Relações com as disciplinas cursadas

As disciplinas do curso que tiveram uma maior relevância para o trabalho foram:

MAE0228 - Noções de Probabilidade e Processos Estocásticos. Ironicamente, foi a única disciplina que eu acabei fazendo duas vezes e foi a mais importante, dado que toda a base do TCC é o Modelo Oculto de Markov, que nada mais é que um modelo duplamente estocástico.

MAC0425 - Inteligência Artificial. Apesar de utilizar muita estatística no TCC, isto se deve ao fato de que o problema que queria resolver era um problema de IA, que é um campo de estudo fortemente baseado nela.

MAC0338 - Análise de Algoritmos. Conhecer diferentes algoritmos para resolver diferentes problemas eficientemente: é o que de modo geral fazemos todo o tempo e por isso considero esta disciplina a mais importante da grade curricular.

MAC0300 - Métodos Numéricos da Álgebra Linear. Todos os algoritmos aqui implementados não teriam utilidade se não soubesse como

tratar corretamente os problemas de *underflow*, arredondamento e erro, que aconteceram bastante.

MAC0211 - Laboratório de Programação I e MAC0242 - Laboratório de Programação II. Com elas que tive a oportunidade de desenvolver um projeto de software do início ao fim, ao longo de todo um semestre, e a aprender mais sobre este processo como um todo.

MAC0342 - Laboratório de Programação Extrema. O mais próximo da realidade fora da Universidade que chegamos no meio acadêmico. Aqui aprendi a fazer software de qualidade com desenvolvimento ágil. Apliquei essa metodologia ao longo do ano.

MAC0110 - Introdução à Computação, MAC0122 - Princípios de Desenvolvimento de Algoritmos, MAC0323 - Estruturas de Dados, MAE0121 - Introdução à Probabilidade e à Estatística I e MAE0212 - Introdução à Probabilidade e à Estatística II. Sem o conhecimento adquirido com as disciplinas introdutórias, nada acima seria possível.

3 Trabalhos Futuros

Foi uma felicidade muito grande ver este trabalho realizado. O software mais interessante que produzi sozinho durante estes 4 anos de BCC. Quando eu entrei no IME, a minha motivação era aprender a programar para fazer jogos. Meu pensamento mudou desde então, mas, coincidentemente, as duas áreas que mais me atraem na computação continuam sendo Inteligência Artificial e Computação Gráfica.

Pretendo continuar estudando essas duas e me pós-graduar, para poder participar do desenvolvimento de softwares que direcionem toda essa extensa teoria já criada para uso prático.

Referências

[1] M. Elmezain, A. Al-Hamadi, and B. Michaelis. Hand gesture recognition based on combined features extraction. In *International Conference on Machine Vision, Image Processing, and Pattern Analysis, PWASET*, volume 60, pages 459–464, 2009.

[2] G. Hamerly and C. Elkan. Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 600–607. ACM, 2002.

[3] T. Leyvand, C. Meekhof, Y.C. Wei, J. Sun, and B. Guo. Kinect identity: Technology and experience. *Computer*, 44(4):94–96, 2011.

[4] N. Liu and B.C. Lovell. Gesture classification using hidden markov models and viterbi path counting. In *The Seventh Biennial Australian Pattern Recognition Society Conference*, volume 1, pages 273–282. Csiro, 2011.

[5] VM Mantyla. Discrete hidden markov models with application to isolated user-dependent hand gesture recognition. *VTT publications*, 2001.

[6] N. M. Oliver. *Towards Perceptual Intelligence: Statistical Modeling of Human Individual and Interactive Behaviors*. PhD thesis, Massachusetts Institute of Technology, 2000.

[7] F. B. Petroni. Grmediator: uma biblioteca modular para reconhecimento de gestos. Master's thesis, Universidade de São Paulo, 2010.

[8] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[9] T. Schlomer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14. ACM, 2008.

[10] J. Sheng. A study of adaboost in 3d gesture recognition. Technical report, University of Toronto, 2005.

[11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of the IEEE International*

Conference on Computer Vision and Pattern Recognition (CVPR'11), Colorado Springs, USA, 2011.

[12] T.E. Starner. Visual recognition of american sign language using hidden markov models. Master's thesis, Massachusetts Institute of Technology, 1995.

[13] D.O. Tanguay Jr. Hidden markov models for gesture recognition. Master's thesis, Massachusetts Institute of Technology, 1995.

[14] L. R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 2003.

[15] T. Yang and Y. Xu. Hidden markov model for gesture recognition. 1994.