GRMediator: uma biblioteca modular para reconhecimento de gestos

Fernando Bertolli Petroni

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. Flávio Soares Correa da Silva

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CNPq

São Paulo, fevereiro de 2010

Agradecimentos

Agradeço primeiramente à minha família, que sempre me estimulou a estudar e me permitiu chegar aonde estou hoje, e quem sabe, muito além. O papel de todos foi essencial para que este trabalho desse frutos. Em particular, agradeço a meus pais e avós por tudo que fizeram por mim até hoje, e à minha tia Regina, por gastar seu tempo revisando o texto comigo e não permitir que eu desanimasse.

Agradeço também a todos meus amigos. A alguns por sua ajuda nos assuntos acadêmicos, a outros por me apoiar nos momentos difíceis e me fazer aprender mais sobre a amizade e a vida. Em particular, agradeço a Pedro Henrique Simões de Oliveira por suas inúmeras contribuições ao trabalho, e aos colegas do LIDET por suas análises, críticas e sugestões de melhora.

Agradeço ao meu orientador Flávio por me ensinar muito nos primeiros passos de minha produção científica, especialmente sobre o que compõe um projeto de pesquisa. Agredeço também à CNPq pelo auxílio financeiro, à Sun por fornecer equipamentos para os experimentos aqui feitos, e à banca de defesa por suas sugestões para a melhora da qualidade deste trabalho.

Resumo

Conforme os computadores existentes evoluem e tornam-se capazes de lidar com tarefas maiores e mais complexas, também surgem novas formas de interação que permitem aos usuários ter maior controle e precisão no uso desses computadores. Com o surgimento de certos tipos de dispositivo, como luvas com detectores de movimento e câmeras, nasceu a área de Reconhecimento de Gestos, cujo objetivo é permitir que usemos linguagem corporal para interagir com computadores, de modo similar ao que fazemos para nos comunicar.

Analisando a metodologia de trabalhos existentes, podemos perceber que um problema de reconhecimento de gestos pode ser dividido de uma forma geral em duas etapas: a *obtenção* dos dados provenientes do dispositivo sendo utilizado e a *classificação* desses dados em gestos. As implementações desses trabalhos efetuam a etapa de obtenção com um dado dispositivo e a etapa de classificação com um dado modelo matemático de classificação. Se houver a necessidade de trocar o dispositivo de entrada ou o modelo matemático de classificação, é necessário também alterar a interface de comunicação, ocasionando a reprogramação de boa parte do sistema.

Este trabalho tem como objetivo o estudo e a criação de uma interface de comunicação entre dispositivo e modelo matemático que permita a livre troca de ambos sem a necessidade de reprogramação do sistema. Um programa de reconhecimento de gestos que use a interface pode ser particionado em componentes intercambiáveis, denominados módulos. Um módulo de dispositivo ou de reconhecimento que respeite essa interface pode ser trocado por outro durante a execução do programa, e o reconhecimento de gestos ainda pode ser efetuado.

É apresentada uma breve introdução ao tema de reconhecimento de gestos e engenharia de componentes. Depois é exposta a proposta para se atingir os objetivos, e o passo-a-passo do raciocínio que levou à construção da arquitetura resultante. São mostrados os detalhes de como foram implementadas a arquitetura, o programa principal e os módulos componentes. São exibidos os testes e experimentos que validam se a arquitetura atingiu o objetivo proposto, e se houve ganho ou perda na acuidade de reconhecimento com o uso desta abordagem. O texto é encerrado com as conclusões obtidas e com uma série de possíveis próximos passos para esta pesquisa.

Palavras-chave: Reconhecimento de Gestos, Engenharia de Componentes, Independência de Dispositivo, Independência de Modelo.

Abstract

As computers evolve and become able to deal with larger and more complex tasks, new forms of interaction are created, which allow users to have greater control and precision in the use of those computers. With the invention of devices such as motion-sensing gloves and cameras, the area of Gesture Recognition was started, whose objective is to allow users to employ body language to interact with computers, much like what humans do to communicate with each other.

A gesture recognition problem can be divided in two steps: the gathering of the data from the device being used, and the classification of the data in gestures. The implementations in existing works perform the gathering step with a given device and the classification step with a given classification mathematical model. If we need to change the input device or the model, we also have to change the communication interface between them, which demands us to reprogram a great portion of the system.

The goal in the present work is the study and creation of a communication interface between input device and mathematical model which allows the free exchange of both without the need of reprogramming the system. A gesture recognition program that uses the interface can be partitioned in interchangeable components, called modules. A device or model module that respects this interface can be exchanged by another during the execution of the program, and gesture recognition can still be performed.

A brief introduction to gesture recognition and component engineering is presented. Afterwards, the proposition to achieve the goals and the step-by-step of the logic that generated the resulting architecture are exposed. We show the details on how were implemented the architecture, the main program and the component modules. We then present the tests and experiments that validate the architecture built this project, and whether there was gain or loss in the recognition accuracy while using this approach. The text ends with the obtained conclusions and with a series of next steps for this research.

Keywords: Gesture Recognition, Component Engineering, Device Independence, Model Independence.

Sumário

Lista de Abreviaturas

HMM	Modelo Oculto de Markov (*Hidden Markov Model*).
FFNN	Feed-Forward Neural Network.
RNN	Rede Neural Recorrente (*Recurrent Neural Network*).
TDNN	Rede Neural de Atraso Temporal (*Time-Delay Neural Network*).

Lista de Figuras

Capítulo 1

Introdução

Conforme os computadores existentes evoluem e tornam-se capazes de lidar com tarefas maiores e mais complexas, também surgem novas formas de interação que permitem aos usuários ter maior controle e precisão no uso desses computadores. Com o surgimento de certos tipos de dispositivo, como luvas com detectores de movimento e câmeras, nasceu a área de Reconhecimento de Gestos, cujo objetivo é permitir que usemos linguagem corporal para interagir com computadores, de modo similar ao que fazemos para nos comunicar.

Trabalhos que fazem referência ao reconhecimento de gestos dividem-se em duas frentes, onde cada frente usa um tipo de dispositivo diferente para atingir o objetivo de reconhecer gestos. Uma das frentes utiliza dispositivos que são anexados a alguma parte do corpo (chamados de dispositivos *intrusivos*) e transmitem ao computador informações relativas à posição ou orientação daquela parte. Conforme o usuário se movimenta, o dispositivo emite constantemente essas informações, que são coletadas e analisadas pelo computador.

Existem vários trabalhos que usam dispositivos intrusivos no reconhecimento. Schlömer *et.al.* [SPHB08], Sreedharan *et.al.* [SZP07], Borza [Bor08] e Kratz *et.al.* [KSL07] usam os dados provenientes do acelerômetro instalado no Nintendo Wiimote [wii] para estimar a trajetória da mão do usuário durante a execução de um gesto. Harling [Har93], Sandberg [San97] e Murakami e Taguchi [MT91] são exemplos de trabalhos que usam luvas que enviam dados relativos à posição ou movimentação da mão do usuário.

A segunda frente depende de câmeras posicionadas no ambiente onde o usuário se encontra (chamados de dispositivos *não-intrusivos*). Esses dispositivos monitoram o usuário, tirando uma série de fotos conforme ele se movimenta. As fotos são analisadas pelo computador, que descobre onde ficam em cada imagem as partes do corpo relevantes para a interpretação do gesto, e posteri-

ormente as diferenças entre as imagens são utilizadas para se determinar qual foi o gesto efetuado. Dentre os trabalhos que utilizam tais dispositivos, podemos citar Cabral *et.al.* [CMZ05], Eickeler *et.al.* [EKR98], Freeman e Roth [FR94] e Yang e Ahuja [YA99].

Com os recentes avanços nesta área, surgiram muitos trabalhos cujo objetivo é permitir que o reconhecimento de gestos seja usado em várias circunstâncias. Borza [Bor08] usa os resultados de seu trabalho para controlar um reprodutor de mídia através de gestos, permitindo que usuários troquem de música, aumentem ou reduzam o volume, e pausem ou resumam a reprodução à distância. Kratz *et.al.* [KSL07] criaram um jogo que utiliza os gestos como principal meio de interação com o jogador. Cabral *et.al.* [CMZ05] construíram seu sistema de reconhecimento de gestos com foco em interatividade em ambientes de realidade virtual, permitindo usar gestos para selecionar objetos, navegar e visualizar o ambiente. Yang e Ahuja [YA99] aplicam o método proposto em seu trabalho no reconhecimento de gestos provenientes da Linguagem Americana de Sinais (ASL). Com isto vemos que cada gesto pode ativar determinadas funcionalidades de um programa.

Analisando a metodologia dos trabalhos existentes, podemos perceber que um problema de reconhecimento de gestos pode ser dividido de uma forma geral em duas etapas: a *obtenção* dos dados provenientes do dispositivo sendo utilizado e a *classificação* desses dados em gestos. A etapa de obtenção consiste em efetuar comunicação com o dispositivo, acumular dados até que o conjunto resultante torne-se classificável, aplicar filtragens e transformar os dados no formato que o classificador espera recebê-los. A etapa de classificação consiste em receber o conjunto de dados e utilizar um *modelo matemático* para decidir a qual dos gestos previamente registrados no sistema esse conjunto se assemelha mais.

A biblioteca Wiigee, produto do trabalho de Schlömer *et.al.* [SPHB08], usa tal divisão. A comunicação com o dispositivo e o pré-processamento (modificações nos dados para ter um reconhecimento mais preciso ou eficiente) são primeiro efetuados, para em seguida os dados resultantes serem utilizados na etapa de classificação com o modelo matemático escolhido no trabalho (HMM). Eickeler *et.al.* [EKR98] têm um sistema baseado em visão computacional, composto de três módulos: pré-processamento, extração de características (*feature extraction*) e classificação. A extração de características consiste em extrair as informações relevantes de mudança de posição para enviar somente essas informações para o classificador, ou seja, é uma espécie de transformação dos dados para posterior classificação; assim, podemos dizer que os dois primeiros módulos são acionados na etapa de obtenção, e o último na etapa de classificação.

As implementações dos trabalhos existentes efetuam a etapa de obtenção com um dado dispositivo e a etapa de classificação com um dado modelo matemático. Isso significa que uma interface

de comunicação entre o dispositivo e o modelo é estabelecida e fixada. Essa interface é criada de modo a satisfazer as necessidades do par dispositivo / modelo escolhido. Entretanto, se houver a necessidade de trocar o dispositivo de entrada ou o modelo matemático de classificação, é necessário também alterar a interface de comunicação, ocasionando a reprogramação de boa parte do sistema. Se desejarmos, por exemplo, trocar uma luva por um dispositivo com acelerômetro, seremos obrigados a modificar toda a parte do sistema responsável por obter os dados da luva para que obtenha os dados a partir do novo dispositivo, e além disso teremos que mudar também a parte que processa e classifica os dados em gestos, porque os dados estarão em um formato diferente do esperado. Um problema semelhante ocorre se desejarmos usar um modelo matemático diferente para classificar os gestos; é necessário reprogramar o sistema e criar uma nova versão que utilize o novo modelo.

Se fosse possível a abstração do tipo de dispositivo e do modelo matemático utilizados, pelo estabelecimento de uma interface fixa entre dispositivo e modelo, qualquer "pedaço" de programa que estivesse de acordo com a interface poderia agir como um coletor de dados de dispositivo ou como modelo matemático de classificação. Esses "pedaços", doravante denominados *módulos*, teriam implementação livre e seriam independentes entre si, o que permitiria a troca de dispositivo e modelo sem reprogramação do sistema.

Este trabalho tem como objetivo o estudo e a criação dessa interface, e de uma biblioteca de reconhecimento de gestos que utilize essa interface e módulos de dispositivo e reconhecimento. A biblioteca deve permitir a troca dos módulos, mudando a fonte de dados e / ou o modelo matemático sem a necessidade de reprogramação do sistema. São também inferidas as restrições que devem ser impostas sobre o processo de reconhecimento de gestos para poder usar a interface em questão, com foco de estudo em dispositivos intrusivos (embora não haja evidência que os resultados deste trabalho não sejam aplicáveis também a dispositivos extrusivos). Não se propõe a criação de métodos mais eficientes de treinamento ou classificação de gestos. De fato, com a arquitetura genérica alguma acuidade de reconhecimento é perdida, o que torna soluções específicas mais eficientes nesse quesito.

No capítulo 1 é apresentada uma breve introdução ao tema de reconhecimento de gestos, os objetivos do trabalho contextualizados e a argumentação que explica porquê são desejáveis trabalhos que aprofundem o conhecimento nesta área, e também porquê as características aqui propostas como objetivo são interessantes. No capítulo 2 são mostrados os princípios teóricos sobre os quais este trabalho se baseia, tanto da área de Reconhecimento de Gestos (seção 2.1) quanto de Engenharia de Software (seção 2.2), e são citados e comparados alguns trabalhos de iniciativa similar a esta (seção 2.3). No capítulo 3 primeiro é exposta a proposta para se atingir os objetivos, e o passo-a-passo do raciocínio que levou à construção da arquitetura resultante (seção 3.1). Em seguida

são levadas em consideração algumas restrições práticas do reconhecimento de gestos para efetuar alterações funcionais na arquitetura (seção 3.2). Finalmente, são mostrados os detalhes de como foram implementadas a arquitetura, o programa principal e os módulos componentes (seção 3.3). No capítulo 4 são exibidos os testes e experimentos que validam se a arquitetura atingiu o objetivo proposto, e se houve ganho ou perda na acuidade de reconhecimento com o uso desta abordagem. O capítulo 5 encerra o texto com as conclusões obtidas e enumera uma série de possíveis próximos passos para esta pesquisa.

1.1 Motivação

A principal motivação do reconhecimento de gestos é permitir uma interação mais agradável e eficiente com o computador ao se efetuar uma determinada tarefa. Nesta seção, discutiremos as vantagens de usar um programa de reconhecimento de gestos que permita a troca de dispositivo e de modelo matemático.

A capacidade de trocar dispositivos beneficiaria principalmente aos usuários finais do programa que utiliza a biblioteca de reconhecimento. A *preferência* do usuário pode ser levada em conta ao desenvolver um programa que reconhece gestos, pois alguns usuários podem sentir-se mais confortáveis usando certo dispositivo, enquanto outros podem preferir um diferente. Se o programa oferece apenas um dispositivo para reconhecimento, somente um grupo de usuários é favorecido. Outro fator importante é a *disponibilidade* dos dispositivos. É possível que um usuário deseje usar um programa, mas dentre os dispositivos que ele possui não há o necessário para a tarefa; neste caso seria interessante existir um recurso que permitisse que fossem usados os seus outros dispositivos.

O programa deve normalmente ser reescrito e preparado para trabalhar com quaisquer opções extras de dispositivo. Tal preparação pode ser bastante custosa, principalmente se o dispositivo atual e o modelo matemático estiverem muito dependentes um do outro. A biblioteca proposta neste trabalho estabelece uma interface genérica de comunicação entre dispositivo e modelo matemático; ao usá-la, a dependência fica bastante reduzida, e pode-se usar outro dispositivo simplesmente trocando de módulo. Neste aspecto, os desenvolvedores também são beneficiados pela *simplicidade* de uso e pela possibilidade de *reutilização* de módulos já desenvolvidos.

A capacidade de trocar modelos matemáticos é mais interessante para os desenvolvedores, pois ao criar um programa eles escolhem o conjunto de gestos que será usado para ativar suas funcionalidades. Dependendo da natureza dos gestos, é possível que um determinado modelo matemático seja mais adequado do que outros. Normalmente os modelos matemáticos são previamente estudados

e só é implementado aquele que oferece o maior potencial teórico de adequação aos gestos usados no programa. Se for usada a biblioteca proposta, os modelos matemáticos podem ser livremente trocados, então se estiverem disponíveis os módulos de vários modelos os desenvolvedores têm a oportunidade de testar facilmente qual deles é mais adequado para o seu caso.

O público-alvo deste trabalho são os desenvolvedores de programas cujas funcionalidades não sejam atreladas a um dispositivo em específico. Isso inclui, por exemplo, reprodutores de mídia, jogos simples e mundos virtuais tridimensionais. Exclui editores de texto e programas de chat, por serem atrelados à digitação, ou programas de desenho e edição de imagens, por serem atrelados à criação e seleção de elementos com o uso do mouse. Com a biblioteca aqui apresentada, o desenvolvedor pode utilizar gestos como uma forma alternativa de ativar funcionalidades de seu programa, tornando a experiência do usuário mais intuitiva e natural, ou fornecendo acessibilidade a usuários com capacidades reduzidas de interação com computadores.

1.2 Objetivos

Os principais objetivos deste trabalho são:

- desenvolver uma interface de comunicação entre componentes que permita que um sistema de reconhecimento de gestos que a utiliza seja, no maior grau possível, independente de dispositivo de entrada e do modelo matemático de reconhecimento.

- especificar as operações que podem ser invocadas em cada tipo de módulo (de dispositivo e de reconhecimento), e o formato dos dados que transitarão entre esses módulos nesta interface, de tal forma que seja possível a troca de módulos sem a necessidade de reprogramação do sistema.

Um objetivo secundário deste trabalho é verificar o impacto da interface sobre a acuidade de reconhecimento.

Capítulo 2

Fundamentação e Teoria

2.1 Reconhecimento de Gestos

Gestos são movimentos feitos com partes do corpo, usados como uma forma alternativa ou complementar de comunicação. É importante ressaltar a diferença entre gestos e *posturas*, que são caracterizadas por uma configuração fixa das partes do corpo em um determinado instante. Gestos consistem de múltiplas configurações das partes do corpo que variam com o tempo. Portanto, posturas são instantâneas, e gestos não.

Para ser capaz de classificar dados provenientes do dispositivo em gestos, o sistema deve ser capaz de guardar informações sobre os gestos que pode reconhecer. Essas informações podem ser *imutáveis*, determinadas durante o desenvolvimento do sistema, ou *mutáveis*, podendo ser fornecidas ou alteradas durante a execução do sistema. Para poder reconhecer gestos diferentes dos atualmente registrados, um sistema imutável precisa ser reprogramado, enquanto um sistema mutável possui um mecanismo que possibilita essa expansão de informações. Esse mecanismo é chamado de *treinamento*, e permite que o sistema seja "ensinado" a reconhecer novos gestos. Este trabalho aplica-se exclusivamente a sistemas mutáveis, ou *treináveis*.

O mecanismo de treinamento pode variar de sistema para sistema, mas há uma prática bastante comum. Como o movimento humano não é totalmente preciso, existem variações entre execuções de um mesmo gesto. Se o sistema de reconhecimento incorporar dados de somente uma execução de gesto, é bastante provável que ele não seja capaz de reconhecer outra execução ligeiramente diferente do mesmo gesto, por não possuir informações sobre *variações comuns*. Para evitar esse problema e aumentar a capacidade de *generalização* do sistema, são normalmente armazenadas informações de múltiplas execuções do mesmo gesto, na expectativa de que o sistema consiga extrair as informações sobre as variações comuns do gesto e classificá-lo corretamente.

Usualmente um sistema mutável é usado em duas fases: *treinamento*, na qual ele "aprende" a classificar novos gestos, e *classificação*, na qual é apresentado um movimento e ele deve decidir se esse movimento corresponde a um gesto treinado ou não. Essas fases não precisam necessariamente ser separadas; por exemplo, um sistema poderia incorporar novas informações sobre um gesto ao efetuar uma classificação e receber uma resposta do usuário sobre a corretude da classificação. Entretanto, este trabalho foi feito considerando que as duas fases são explicitamente distintas, ou seja, se ele está em treinamento não classifica, e se está classificando não "aprende".

2.1.1 Processamento dos Dados

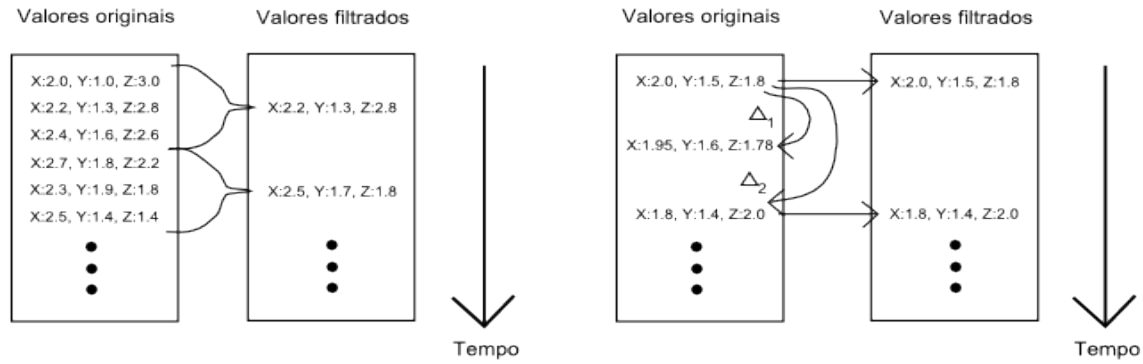
Para que os dados obtidos do dispositivo sejam úteis no reconhecimento de gestos, muitas vezes é necessário um pré-processamento que elimine características indesejáveis e que prepare o conjunto de dados para ser classificado pelo modelo matemático, ou para ser armazenado na etapa de treinamento. Nesta seção são apresentados alguns dos processos frequentemente utilizados.

- Filtragem

  O processo de filtragem procura reduzir a quantidade de informações que é enviada ao modelo matemático. Muitos dispositivos possuem taxas de amostragem elevadas (mais de 100 leituras por segundo), o que gera um conjunto de dados demasiado grande até para gestos curtos. Quanto maior o conjunto de dados, mais tempo o modelo matemático leva para fazer uma classificação, portanto é interessante que dados que não façam diferença na classificação sejam omitidos para melhorar a performance.

  Um exemplo de filtragem consiste em escolher um número P e tirar a média de cada P valores provenientes do dispositivo (como na figura 2.1(a)). Quanto maior a taxa de amostragem, a tendência é haver menos diferenças entre leituras consecutivas (porque não nos movemos rápido o suficiente para tal), e assim pode-se usar valores maiores de P para obter menos dados com mínima perda de informação. Uma boa escolha de valor para P depende do dispositivo usado.

  Outra técnica, usada em [SPHB08], é chamada de *filtro de equivalência direcional* e leva em consideração a variação das informações entre uma leitura e a próxima. Quando a duração de um movimento não é importante para a classificação, e a próxima leitura é muito similar à anterior, pode-se omitir uma das duas, e repetir o processo para as próximas até que seja encontrada uma leitura com variação grande o suficiente, que passa pelo filtro (veja figura 2.1(b)).



(a) Exemplo de filtragem de média com P = 3. (b) Exemplo de filtragem de equivalência direcional.

Figura 2.1: *Ilustração de dois processos usuais de filtragem.*

Ainda no caso que a duração do movimento é desconsiderada, pode-se ainda ignorar leituras de magnitude desprezível, abaixo de um determinado limiar (*threshold*). Essa técnica é usada em [SPHB08] para desconsiderar os momentos de repouso do dispositivo, registrando somente os movimentos de intensidade relevante.

- Quantização

  O processo de quantização tem como objetivo reduzir a quantidade de possíveis valores provenientes do conjunto de dados. Alguns modelos matemáticos não são capazes de lidar com a amplitude dos valores puros fornecidos pelo dispositivo; outros até podem funcionar, mas com um custo de performance elevado. A quantização divide o espaço de possíveis valores de leituras do dispositivo em um número finito de possibilidades, e mapeia cada valor lido em um *codebook vector*, sendo que um *codebook* é uma espécie de tabela de referência. Esse processo pode ser encarado como compressão de dados com perda de informação.

  Consideremos o caso de um vetor de aceleração V no espaço tridimensional. Ele possui componentes que têm valores nos eixos X, Y e Z, refletindo a variação da velocidade do dispositivo em cada eixo.

$$V = (x, y, z) \tag{2.1}$$

  Como são valores numéricos, há infinitas possibilidades de combinações das três variáveis, e alguns modelos matemáticos precisam de uma limitação na quantidade de valores distintos recebidos como entrada. Uma alternativa para reduzir as possibilidades é escolher N vetores do espaço tridimensional:

$$V_1 = (x_1, y_1, z_1), V_2 = (x_2, y_2, z_2), ..., V_N = (x_N, y_N, z_N) \tag{2.2}$$

Para cada um dos N vetores, calculamos a distância ao vetor V:

$$D_{V_i,V} = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \tag{2.3}$$

O vetor V_i que tiver a menor distância até V é escolhido para ser seu representante; em outras palavras, V é mapeado em V_i, e há somente N possibilidades de valores provenientes deste processo. Para haver o mínimo de perda de informação, é importante escolher um conjunto de vetores que minimize a esperança da distância entre um vetor qualquer e os vetores do conjunto. Este procedimento é efetuado em [SPHB08], com a escolha de 14 vetores distribuídos pelo espaço tridimensional e escolhidos de forma a estarem o mais próximo possível de qualquer vetor de aceleração.

2.1.2 Modelo Matemático

O termo "modelo matemático" é usado neste trabalho significando qualquer estrutura de dados sobre a qual possam ser aplicados algoritmos para armazenar informações sobre gestos e para classificar uma sequência de dados em um gesto. Esta definição genérica permite que quase qualquer estrutura de dados seja considerada um modelo matemático, desde que haja algoritmos capazes de operar sobre ela. Entretanto, para ser um modelo matemático *eficiente*, algumas propriedades são desejáveis. Alguns exemplos:

- a estrutura deve ser capaz de guardar informações temporais. Como gestos são movimentos ao longo do tempo, uma estrutura que sirva para capturar informações instantâneas não funcionará bem.
- o algoritmo de classificação deve ser rápido o suficiente para que o usuário do sistema não perceba um atraso substancial entre o fim da execução de um gesto e a resposta do sistema. (reação rápida)
- dada uma sequência de dados que não corresponde bem a nenhum dos gestos treinados, uma classificação com essa sequência deveria gerar uma resposta nula ao invés de responder com o gesto mais provável (tal sistema é chamado de ávido, ou *eager system*)

Serão apresentados exemplos do que é considerado um modelo matemático para este trabalho.

Modelos Ocultos de Markov (Hidden Markov Models)

Um Modelo Oculto de Markov (HMM) é um modelo estatístico que tenta estimar o comportamento de um fenômeno baseado somente nas observações provenientes da evolução do estado do fenômeno no decorrer do tempo, sem ter acesso ao estado do fenômeno em si.

Um exemplo de situação que pode ser modelada com um HMM é o seguinte: suponha que o dia pode estar ensolarado, nublado ou chuvoso. Estamos fechados em um lugar que não permite descobrir como está o tempo lá fora. Queremos descobrir como está o dia, mas o único indicador de que dispomos é um medidor de umidade do ar, que nos diz se o ar está seco, normal ou úmido. A umidade do ar, sozinha, não nos indica com certeza como está o dia, mas índices altos de umidade indicam uma alta probabilidade de estar chovendo. A partir de observações provenientes de um estado (a umidade associada ao clima do dia), podemos estimar a probabilidade do clima ser ensolarado, nublado ou chuvoso (estamos estimando o comportamento do clima no dia).

Para definir um HMM, portanto, é necessário o seguinte conjunto de parâmetros([Rab89]):

- o número N de possíveis estados. Em nosso caso, teríamos 3 possíveis estados para o clima: ensolarado, nublado ou chuvoso. Eles definem o conjunto de estados $S = \{S_1, S_2, ..., S_N\}$.

- o número M de observações emitíveis por estado. Em nosso caso também teríamos 3 possíveis observações: seco, normal e úmido. Elas definem o conjunto de observações $V = \{V_1, V_2, ..., V_M\}$.

- a distribuição de probabilidade $A = \{a_{ij}\}$, onde $a_{ij} = P[q_{t+1} = S_j | q_t, = S_i], 1 \leq i, j \leq N$.

  Se estivermos em um estado i qualquer no instante t, essa distribuição nos mostra a probabilidade de mudarmos para o estado j no próximo instante. Em nosso caso ela nos diz qual é a chance do clima estar chuvoso no dia seguinte, dado que hoje estava ensolarado, por exemplo.

- a distribuição de probabilidade $B = \{b_j(k)\}$, onde $b_j(k) = P[v_k \; em \; t | q_t = S_j], 1 \leq j \leq N, 1 \leq k \leq M$.

  Se estivermos em um estado j qualquer no instante t, essa distribuição nos mostra a probabilidade de obtermos o valor de observação k naquele estado. Em nosso caso ela nos diz qual é a chance do medidor acusar ar úmido dado que hoje está ensolarado, por exemplo.

- a distribuição de estado inicial $\pi = \{\pi_i\}$, onde $\pi_i = P[q_1 = S_i], 1 \leq i \leq N$.

  Essa distribuição nos diz a chance de começarmos em cada um dos estados.

Usamos a notação simplificada $\lambda = (A, B, \pi)$ para nos referir ao conjunto completo de parâmetros do modelo, já que podemos inferir N e M a partir das dimensões das distribuições de probabilidade A e B.

HMMs foram explorados em um número de trabalhos para uso em reconhecimento de gestos. Em muitos deles ([SPHB08], [Män01], [EKR98], [Bor08] e [KSL07]) os estados do HMM correspondem a estágios dos gestos, e as observações do HMM correspondem a transformações dos dados obtidos a partir dos dispositivos de entrada. Como os estágios não são explicitamente definidos, são considerados ocultos, assim como os estados no HMM, e a única coisa que dá alguma informação sobre o estágio do gesto em que o usuário se encontra são as observações. Cada observação tem uma chance de acontecer em cada estágio do gesto.

Surgem pontos importantes a resolver usando esta abordagem. Quantos estados o HMM deve ter? Quantas possíveis observações distintas? Quais as probabilidades das transições entre estados? Como se determina a probabilidade da emissão de uma observação em um dado estado? Não existem respostas ótimas para estas perguntas; cada trabalho usa uma estratégia para determinar suas respostas. A implementação de HMMs usada neste trabalho foi baseada em [SPHB08], por isso são adotadas as mesmas estratégias.

O número de estados do HMM é fixo (8) e determinado empiricamente. As observações distintas eram originalmente fixadas no número máximo de valores distintos gerados a partir da quantização dos dados provenientes do Nintendo Wiimote (no caso, 14), mas isso foi modificado para que o HMM pudesse ser usado em conjunto com outros dispositivos e outras quantizações; o número de observações distintas é determinado na etapa de treinamento.

Um gesto é uma série de movimentos que evolui ao longo do tempo. Se cada estado representa um estágio do gesto, espera-se que o gesto comece em um determinado estado e transite pelos estados intermediários, um a um, até chegar no estado final. Da mesma forma, não são esperados "saltos" grandes entre os estados, pois eles implicariam na quebra do fluxo do gesto. Por isso foi adotado um modelo de HMM chamado de *left-to-right* (ou modelo de *Bakis* [Rab89]), que possui as seguintes características:

$$a_{ij} = 0,\ j < i \ ou\ j > i + \Delta$$

$$\pi_i = \begin{cases} 0 & , i \neq 1 \\ 1 & , i = 1 \end{cases} \tag{2.4}$$

Este modelo garante que o gesto sempre comece em seu estado inicial e, em cada passo, perma-

neça no mesmo estado ou avance no máximo Δ estados.

A última pergunta tem sua resposta nos algoritmos de treinamento dos HMMs. Um HMM sem nenhuma informação tem, a princípio, chances iguais de obter todas as observações em todos os estados. Conforme o modelo é treinado, essas probabilidades são alteradas de acordo com as informações que ele incorpora. Na seção 2.1.3 é mostrado um algoritmo de treinamento.

Redes Neurais (Neural Networks)

Redes Neurais são estruturas de dados compostas de inúmeras sub-estruturas ligadas entre si, tal como os neurônios de um cérebro. Uma rede neural possui um conjunto de entradas e um conjunto de saídas. Um estímulo aplicado sobre as entradas é transmitido para uma camada de neurônios, que altera e repassa esse estímulo para todos os neurônios da próxima camada ligados a eles, e o processo se repete até que os neurônios do conjunto de saídas sejam ativados, gerando a resposta da rede ao estímulo. Em termos computacionais, uma rede neural pode ser comparada a um grafo direcionado no qual os neurônios são nós e as arestas direcionadas com pesos são as conexões entre as saídas de um neurônio e as entradas de outro.[JMM96]

Um neurônio soma o valor de seus n estímulos de entrada e aplica uma função no resultado, gerando um valor de saída:

$$y = \theta \left(\sum_{j=1}^{n} w_j x_j \right) \tag{2.5}$$

onde os w_j são os pesos de cada conexão de entrada ao neurônio, x_j o estímulo recebido na conexão de entrada e θ a função aplicada. A saída do neurônio é então usada como a entrada de outros neurônios, até que o sinal chegue às saídas da rede.

Os neurônios da rede podem estar interligados de muitas formas, mas a arquitetura da rede pode ser classificada em uma de duas categorias: redes *feed-forward* (ou Feed-Forward Neural Networks, FFNN) ou *recorrentes* (ou Recurrent Neural Networks, RNN). Os neurônios de uma determinada camada de uma FFNN estão sempre ligados somente aos neurônios de camadas posteriores, ou seja, o grafo formado pela rede não possui ciclos. RNNs não possuem essa restrição, podendo efetuar conexões na mesma camada ou em camadas anteriores. FFNNs normalmente são estáticas, sempre apresentando somente um conjunto de saídas como resposta a um conjunto de entrada, enquanto RNNs são dinâmicas, mantendo um estado interno que varia e permite a obtenção de uma sequência de saídas para um conjunto de entrada.[JMM96]

Um ponto crucial sobre as redes neurais é que, dado um determinado estímulo inicial, seus

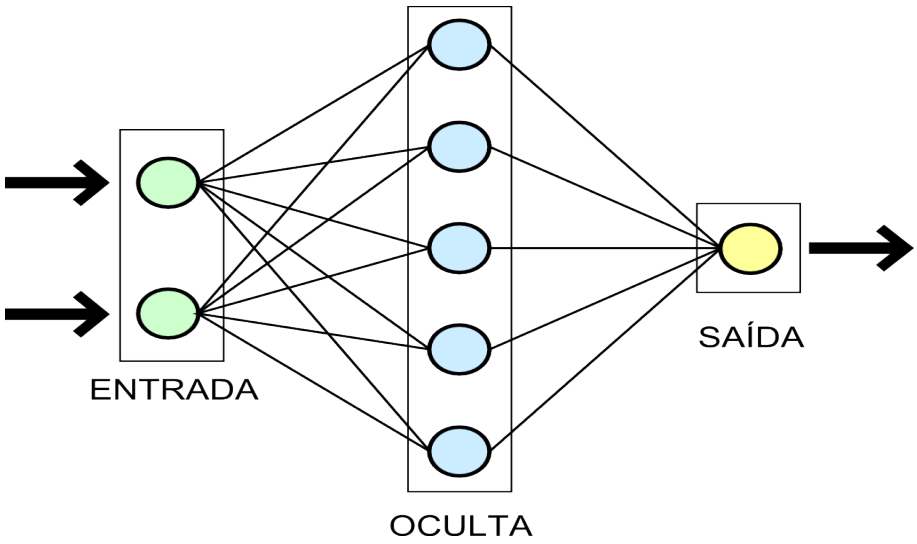


Figura 2.2: *Representação visual de uma rede neural. Valores aplicados na camada de entrada trafegam pela rede até chegar à camada de saída.*

neurônios têm a capacidade de mudar os pesos entre as conexões, e consequentemente o quanto é alterado o estímulo inicial, de forma que a resposta da rede a esse estímulo mude para uma resposta desejada. Em outras palavras, a rede pode ser *treinada* para fornecer uma certa resposta a um estímulo.

A capacidade de alterar o comportamento de uma rede neural faz dela uma boa candidata para resolver uma série de problemas, tais como classificação de padrões, categorização, aproximação de funções, previsões e otimização [JMM96]. Como o reconhecimento de gestos é uma forma específica de classificação de padrões (dada uma sequência de movimentos, enquadrá-la em uma de um número de categorias pré-existentes), usar uma rede neural adaptada para classificar padrões é uma abordagem natural e usada em vários trabalhos (como [Har93], [San97], [MT91] e [YA99]).

Em uma rede neural desse tipo, cada entrada é associada a uma informação proveniente de um movimento, como aceleração, velocidade, posição ou alguma função envolvendo essas variáveis, e cada saída é associada a um dos gestos treinados. Ao alimentar a rede com dados obtidos a partir de um gesto, toda a informação é processada pelas camadas intermediárias de neurônios e as saídas são ativadas, cada uma com uma intensidade diferente, sendo que a saída que apresenta maior valor representa o gesto mais provável a qual o movimento corresponde.

Para aplicar uma rede neural neste contexto, também é necessário determinar uma série de parâmetros. Quantos neurônios deve ter a rede? Quantas entradas e saídas deve a rede ter? Como devem interligar-se (qual arquitetura de rede usar)? Como determinar os pesos entre as conexões de modo que, dada uma entrada que representa um gesto, a rede responda corretamente? Similarmente aos HMMs, não existe uma resposta definitiva a estas perguntas, e cada trabalho responde-as de uma

forma. Normalmente são feitos experimentos modificando parâmetros da rede até que se encontre uma configuração adequada para o problema em mãos.

Variar o número de neurônios nas camadas intermediárias da rede (chamadas também de camadas ocultas, ou *hidden layers*) pode mudar a velocidade de convergência do treinamento e a capacidade de generalização da rede. Mesmo o número de camadas ocultas também faz uma diferença considerável. Neste trabalho foi usada uma rede com uma camada oculta com 15 neurônios, um número bastante pequeno considerando a complexidade do problema de reconhecimento de gestos. O número de entradas da rede será discutido adiante, quando for detalhado o formato de entrada da rede (seção 3.3.8), e o número de saídas é o mesmo que o número de gestos reconhecíveis pela rede.

Para nosso problema, seria interessante que tivéssemos uma rede que levasse em consideração informações obtidas em múltiplos instantes, ou seja, uma rede dinâmica como mencionado anteriormente. Um subtipo de FFNN que aproxima-se bastante desse comportamento são as chamadas Redes Neurais de Atraso Temporal (ou *Time Delay Neural Networks*, TDNN).

Basicamente, uma TDNN é uma FFNN cujos neurônios ocultos e de saída são replicados com o tempo ([Hay05]). Cada neurônio de entrada é ligado a uma sub-camada intermediária de armazenamento. Essa camada tem um determinado número de neurônios (chamados também de *taps*). Cada vez que a rede recebe uma entrada, o novo valor apresentado por cada neurônio é "enfileirado" nessa camada intermediária; a camada oculta então usa os valores dos *taps* para determinar seu próprio valor. O efeito dessa estrutura é que, com o passar do tempo e a apresentação de novas entradas, a rede é capaz de usar valores prévios no cálculo do valor atual, sendo que o número de *taps* representa a quantidade de instantes anteriores que são levados em consideração.

Apesar das TDNNs serem boas candidatas para a tarefa, criá-las e modificá-las corretamente é trabalhoso. Uma FFNN com um número adequado de entradas, que receba todas as informações temporais do gesto de uma só vez, pode ter um desempenho satisfatório sem a complexidade de uma TDNN. Além disso, as TDNNs adicionam um novo parâmetro a ser estimado: o número de *taps*, que deve ter uma relação direta com a duração dos gestos efetuados. Neste trabalho foi usada uma FFNN cuja construção é melhor detalhada na seção 3.3.8.

Finalmente, para determinar os pesos entre as conexões, é usado um algoritmo de treinamento de redes neurais que tenta minimizar a diferença entre a saída apresentada pela rede e a saída esperada. Na seção 2.1.3 é descrito esse algoritmo.

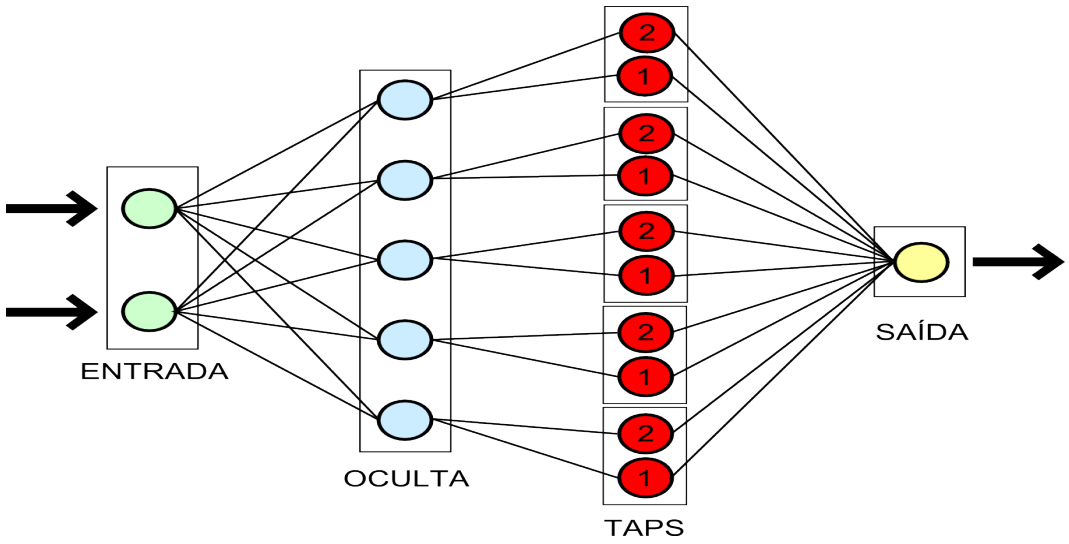


Figura 2.3: *Representação visual de uma TDNN com 2* taps *entre a camada oculta e a de saída. Essa rede também leva em consideração os valores da última execução no cálculo da próxima saída.*

2.1.3 Treinamento

Na fase de treinamento, o sistema deve aplicar algum algoritmo sobre o conjunto de dados e o modelo de modo que este adquira as informações necessárias para reconhecer posteriores sequências de dados semelhantes como instâncias do novo gesto treinado.

Treinamento em HMMs

O exemplo mais notório de treinamento em HMMs é chamado de método de Baum-Welch [Rab89] e consiste em, dados um modelo de HMM $\lambda = (A, B, \pi)$ e uma sequência de observações $O = \{O_1, O_2, ..., O_t\}$, encontrar um modelo $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ tal que a probabilidade de obter O dado $\bar{\lambda}$ (ou $P(O|\bar{\lambda})$) seja localmente maximizada. O procedimento consiste de vários passos, sendo que em cada passo é encontrado um novo modelo $\bar{\lambda}$ tal que:

- $P(O|\bar{\lambda}) = P(O|\lambda)$, e encerramos o processo tendo $\bar{\lambda}$ como resultado, ou
- $P(O|\bar{\lambda}) > P(O|\lambda)$, e repetimos o processo usando $\lambda = \bar{\lambda}$.

Este método é adequado para preparar o modelo dado uma única execução de gesto, mas não é suficiente para múltiplas execuções; é necessário um procedimento que opere sobre um conjunto de sequências, e não descarte a informação obtida para as sequências anteriores ao adaptar o modelo para uma nova. Um método para tal é apresentado em [Män01] e usado neste trabalho.

Treinamento em Redes Neurais

O treinamento de nosso interesse neste trabalho é chamado na literatura de *aprendizado supervisionado*, onde fornecemos à rede a saída esperada para uma determinada entrada. Um dos procedimentos de aprendizado supervisionado usado com frequência é o algoritmo de retro-propagação de erro (*Error back-propagation algorithm* [JMM96]). Esse algoritmo recebe uma conjunto de entradas e a saída correspondente como parâmetro, e consiste dos seguintes passos:

- Alimente a rede com a entrada, obtendo o valor de saída;
- Calcule a diferença entre a saída esperada e a obtida na camada de saída;
- Propague os erros para as camadas anteriores, calculando suas diferenças também;
- Atualize os pesos das ligações entre os neurônios de acordo com o erro;
- Repita a partir do primeiro passo até que o erro esteja abaixo de um limiar específico, ou um determinado número de iterações seja alcançado.

Para várias execuções de gestos, o algoritmo é repetido para cada execução, com o intuito de melhorar a capacidade de generalização da rede.

2.1.4 Classificação

Nesta etapa o modelo matemático recebe uma sequência de dados pré-processados e utiliza um algoritmo para determinar a qual dos gestos já treinados essa sequência mais se assemelha. Normalmente a resposta é um conjunto de probabilidades, sendo que a probabilidade $P(G)$ representa a chance da sequência de dados ser uma instância do gesto G. Posteriormente é aplicado um algoritmo de classificação sobre essas probabilidades para determinar se há alguma probabilidade com valor grande o suficiente para ser considerada como uma classificação.

Classificação em HMMs

A classificação em HMMs convencionais é feita através de um algoritmo chamado de *forward procedure* [Rab89]. Dada uma sequência de observações $O = O_1, O_2, ..., O_T$, desejamos calcular a probabilidade dessa sequência acontecer dentro do modelo λ, ou seja, $P(O|\lambda)$.

Essa probabilidade é definida em função de uma série de variáveis α denominadas *forward variables*:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.6}$$

Cada uma das variáveis representa a probabilidade da sequência até a observação O_T ter sido obtida, e o estado final ser o estado i. O ponto crucial do algoritmo é que cada $\alpha_t(i)$ é definido em função de todos os $\alpha_{t-1}(i)$ anteriores (por indução):

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1,\ 1 \leq j \leq N. \tag{2.7}$$

Com isto, falta apenas definir o valor dos $\alpha_1(i)$, a base da indução. Eles são calculados de acordo com a distribuição de probabilidades dos estados iniciais π:

$$\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N. \tag{2.8}$$

Como cada HMM representa o modelo de um gesto, pode-se aplicar o *forward procedure* em cada um dos HMMs, e o modelo que gerar a maior probabilidade para aquela sequência corresponde ao gesto mais provável.

Classificação em Redes Neurais

Efetuar classificação em uma rede neural onde cada saída representa um gesto é bastante simples. Basta fornecer a sequência de observações provenientes do dispositivo como entrada para a rede, e observar os valores fornecidos na camada de saída. Cada um dos nós será ativado com uma intensidade diferente; o valor apresentado representa a chance que a rede atribuiu de a sequência passada na entrada ser uma execução do gesto representado por aquele nó. Portanto, o maior valor dentre os nós representa o gesto mais provável. Entretanto, se o maior valor for muito pequeno (abaixo de um determinado limiar), pode-se considerar que a sequência não corresponde bem a nenhum dos gestos reconhecíveis pela rede; nesse caso, para reduzir a avidez do modelo, é interessante que a sequência seja rejeitada. Assim, no caso das Redes Neurais, o algoritmo de classificação resume-se à seleção do maior valor e à aplicação do limiar de reconhecimento. Esse limiar pode ser constante e determinado empiricamente, ou variável e determinado em função dos valores obtidos na saída da rede.

2.2 Engenharia de Componentes

A Engenharia de Componentes é uma sub-área da Engenharia de Software que busca, em um dado problema ou sistema a ser desenvolvido, a separação de responsabilidades (*separation of concerns*) entre as partes do sistema, de forma que cada parte seja responsável por resolver um pedaço do problema. Essas partes são denominadas *componentes* do sistema.

Não existe um consenso sobre a definição exata de um componente. Em alguns casos um único objeto faz o papel de um componente, e em outros componentes são pacotes formados por centenas de objetos e recursos. Por exemplo, no Workshop on Component-Oriented Programming [SP03] a seguinte definição de componente foi apresentada:

"Um componente de software é uma unidade de composição com interfaces especificadas contratualmente e somente dependências explícitas de contexto. Um componente de software pode ser implantado independentemente e é sujeito a composição por terceiros."

Esta definição permite várias interpretações, e outros autores usam as suas próprias [Szy02]. Por isso, componentes são melhor definidos pelo que eles oferecem de serviços ao resto do sistema, e de que eles precisam para funcionar adequadamente. Esses aspectos constituem a *interface de uso* do componente.

A interface de uso é logicamente dividida em duas categorias. Informações e pré-condições das quais o componente necessita para funcionar adequadamente são passadas através da interface de requisitos (*required interface*), e as informações provenientes da solução do problema pelo componente e pós-condições são obtidas a partir da interface de fornecimento (*provided interface*). Na prática, as duas interfaces são unidas na mesma interface que é usada tanto pelo componente cliente quanto pelo fornecedor. Com a definição da interface de uso, um componente é capaz de comunicar-se com o resto do sistema, e quem desenvolve as outras partes do sistema não precisa saber os detalhes de implementação do componente, concentrando-se apenas nas interações entre as partes.

No exemplo da figura 2.4 o componente que efetua reservas em feriados utiliza vários outros componentes. Um deles é o componente que efetua reservas em hotéis; os dois componentes comunicam-se através da interface IHotelRes. O próprio componente responsável por efetuar reservas em hotéis usa outros dois componentes, um responsável por gerenciar o programa de lealdade (ofertas obtidas através de certo número de usos do serviço) e outro responsável por efetuar os pagamentos com cartão de crédito.

Com isto, chegamos aos objetivos da Engenharia de Componentes. Um componente é criado para encapsular os detalhes da solução de um problema, fornecendo apenas as operações necessárias

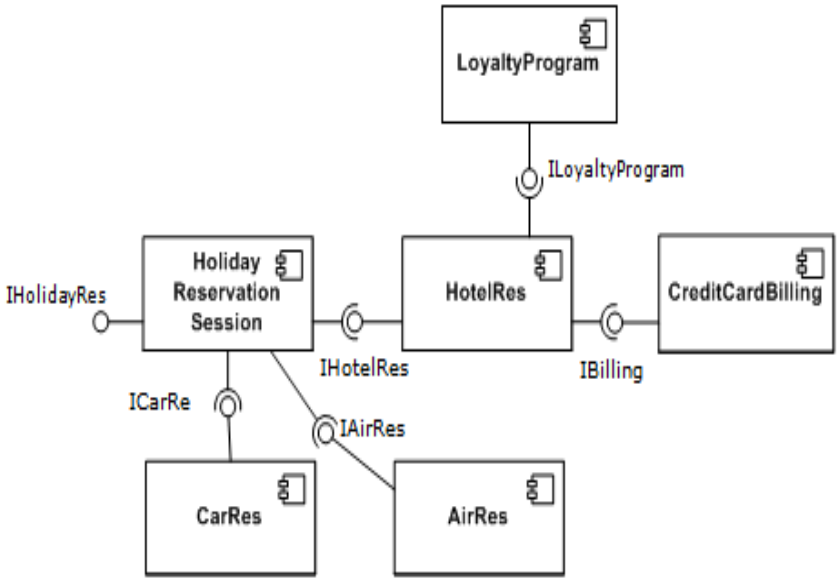


Figura 2.4: *Diagrama de componentes de um sistema que efetua reservas em feriados.*

para defini-lo e resolvê-lo. Usar componentes é muito vantajoso para desenvolvedores de software, por vários motivos:

- a quantidade de detalhes que um desenvolvedor precisa ter em mente ao tentar resolver um problema que depende da saída de um componente é reduzida. O desenvolvedor não precisa saber *como* o componente faz o seu trabalho, somente *o quê* ele faz;
- componentes comunicam-se através de interfaces. Não importa o que esteja atrás da interface se o contrato por ela estabelecido é cumprido; por isso, componentes são substituíveis, e podem ser livremente trocados enquanto satisfizerem a interface de uso;
- se for necessário modificar a forma que um componente resolve o seu problema, o desenvolvedor não precisa de conhecimento dos possíveis clientes do componente, concentrando-se somente na sua solução (enquanto a modificação não implicar em alterações na interface de uso do componente).

Podemos observar as três características no nosso exemplo. Ao fazer modificações na implementação do componente de reservas em feriados, não precisamos nos preocupar com os detalhes relacionados a hotéis, que são gerenciados por seu próprio componente. Podemos trocar a bandeira de cartão de crédito aceita pelo sistema simplesmente escolhendo o componente adequado, se tanto o anterior como o novo satisfizerem a interface IBilling. Finalmente, se for necessário fazer alterações na implementação do componente de reservas em hotéis, não é preciso se preocupar com os possíveis clientes desse componente (como por exemplo o componente de reservas em feriados)

enquanto não houver modificações na interface IHotelRes. São essas três características que motivaram o uso de Engenharia de Componentes neste trabalho, e a interface aqui proposta busca aplicar essa metodologia para obter os benefícios.

2.3 Trabalhos Correlatos

Aqui são listados trabalhos que, além de lidar com reconhecimento de gestos, tentaram alcançar a independência de dispositivo no reconhecimento.

Eisenstein *et.al.* propuseram em seu trabalho [E+03] um arcabouço para reconhecimento de gestos feitos com luvas providas de sensores. O objetivo do arcabouço era possibilitar o uso de qualquer espécie de luva para fazer o reconhecimento através da conversão de dados específicos de cada luva para uma representação semântica da mão humana. Usando luvas providas de menos sensores, o sistema extrai menos informações dos movimentos, mas ainda consegue efetuar classificações usando os dados obtidos. Outra característica considerada importante é a extensibilidade do arcabouço, que permite a incorporação de novos gestos sem a necessidade de reaprender os antigos. Experimentos mostraram grande potencial do arcabouço quando comparado com um sistema que usava uma rede neural convencional. Entretanto, o trabalho concentrou-se apenas em posturas, e a extensão do arcabouço para suportar gestos foi identificada como trabalho futuro.

Faisstnauer *et.al.* [FSS98] buscaram a independência de dispositivo em mundos virtuais com o uso de um módulo denominado Mapper, que serve de mediador entre os dispositivos em uso e a aplicação. O módulo fornece quatro níveis de abstração para os dados provenientes do dispositivo (dados puros, interação, navegação e metáfora), e a aplicação pode escolher qual nível será usado de acordo com a circunstância. O Mapper trabalha com novos dispositivos simplesmente com a adição de um tradutor de dispositivo (*device driver*), ou seja, não é necessário alterar o código da aplicação para que novos dispositivos sejam aceitos.

He e Kaufman [HK93] alcançam o mesmo objetivo de uma forma semelhante com a *Device Unified Interface* (DUI), um protocolo de comunicação entre dispositivos e aplicações. O foco da DUI é facilitar interações em sistemas tridimensionais abstraindo o tipo de dispositivo em uso. Os aplicativos precisam somente interagir com os dados de um dispositivo virtual, que são criados e manipulados pela DUI sem intervenção dos aplicativos. Para usar novos dispositivos basta escrever o tradutor que mapeia as leituras do dispositivo para os dados do dispositivo virtual.

Os dois últimos trabalhos são parecidos e têm vários pontos em comum com este. A principal diferença é que a interface aqui proposta serve para obter informações de qualquer tipo de disposi-

tivo, mas as informações obtidas não são disponibilizadas para a aplicação, e sim para o módulo de reconhecimento, que usa os dados para classificar ou treinar gestos. Mesmo que a aplicação requisitasse esses dados, eles seriam de pouca utilidade para ela, por estar em um formato especialmente criado para a comunicação entre módulos. Em outras palavras, esta abordagem é focada no reconhecimento de gestos, enquanto as outras são focadas em interação e navegação para aplicativos tridimensionais; por isso é bastante inadequado o uso desta abordagem para outros fins. Outra diferença é que esta abordagem também permite a troca de modelo matemático de classificação, um aspecto menos relevante para os outros trabalhos.

Capítulo 3

Metodologia

A proposta para atingir e validar o objetivo do trabalho é composta de três atividades:

- Criação da interface e determinação das restrições que devem ser feitas sobre os dispositivos, modelos e gestos, com base na fundamentação teórica e em trabalhos existentes.
- Criação de uma biblioteca de reconhecimento de gestos que utilize componentes que implementam essa interface.
- Criação de módulos de dispositivo e reconhecimento que implementem a interface para ser usados em conjunto com a biblioteca.

As duas primeiras seções deste capítulo correspondem à primeira atividade.

3.1 Definição da fronteira entre os módulos de dispositivo e modelo matemático

Para poder especificar as operações que poderão ser invocadas em cada tipo de módulo, é necessário definir qual módulo é responsável por qual etapa do processo de treinamento ou reconhecimento. Há duas principais formas de efetuar essa divisão; nesta seção, são apresentados e discutidos as possibilidades e o ponto de divisão escolhido.

Na figura 3.1 os retângulos representam recursos e as elipses representam etapas do reconhecimento de gestos (etapas de um tipo específico de sistema de reconhecimento de padrões [Män01]). Linhas tracejadas mostram as dependências entre tarefas e recursos, e as setas indicam o fluxo de informações entre as etapas.

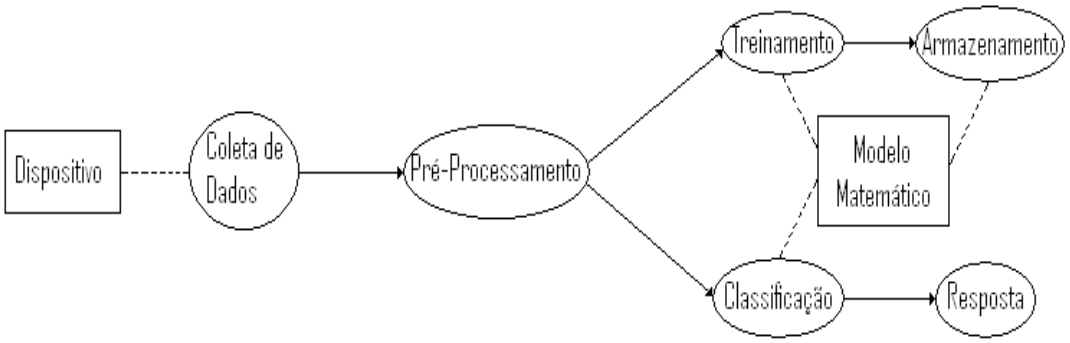


Figura 3.1: *Fluxo de informações e dependências entre etapas do processo de reconhecimento de gestos.*

A etapa de coleta de dados depende de comunicação com o dispositivo. Os dados coletados são usados como entrada na etapa de pré-processamento, que efetua suas operações (filtragens, quantização) e passa os dados resultantes para a etapa de treinamento ou de classificação, dependendo do que se deseje fazer no momento. As duas etapas dependem de acesso ao modelo matemático, e o treinamento gera informações que alteram o estado do modelo matemático, que devem também ser armazenadas. A resposta é obtida a partir do resultado da etapa de classificação.

Definimos que o módulo de dispositivo deve possuir o recurso "dispositivo", e o módulo de reconhecimento deve possuir o recurso "modelo matemático". Além disso, é importante que o módulo de dispositivo não faça referências diretas ao modelo matemático; se for esse o caso, o módulo de dispositivo torna-se dependente de um modelo em particular, e não podemos trocar o módulo de reconhecimento por um que não use aquele modelo específico. Da mesma forma, o módulo de reconhecimento não deve fazer referências diretas ao dispositivo. Com as etapas mencionadas, temos somente duas maneiras de dividir as responsabilidades de cada módulo sem violar as restrições impostas, que são discutidas adiante.

3.1.1 Módulo de dispositivo responsável apenas por obter dados puros do dispositivo

A primeira possibilidade de divisão é fazer módulos de dispositivo que sejam responsáveis somente pela comunicação e obtenção de dados do dispositivo. Assim, o pré-processamento, treinamento, armazenamento e classificação dos dados seriam responsabilidades do módulo de reconhecimento. Observe a figura 3.2.

O módulo de dispositivo seria capaz de:

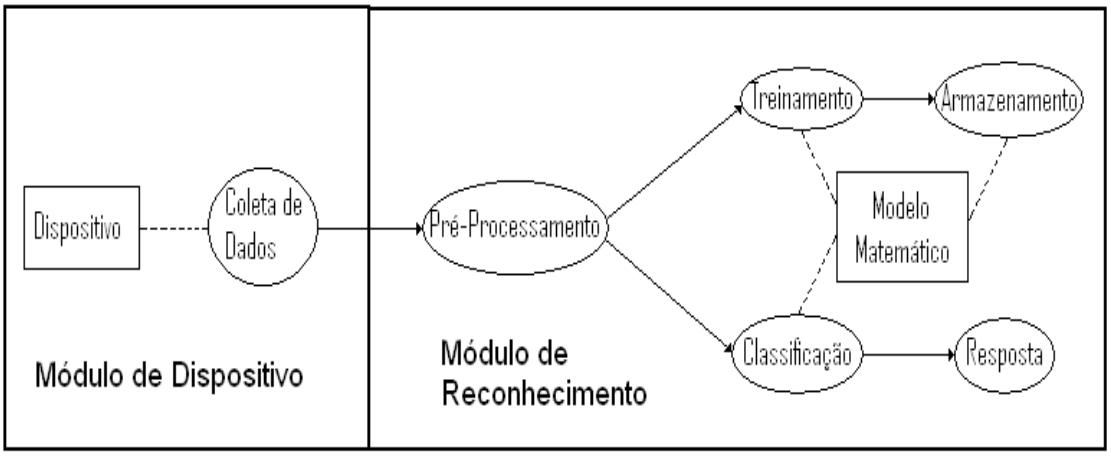


Figura 3.2: *Primeira divisão de responsabilidades entre os módulos.*

- iniciar ou interromper a comunicação com o dispositivo;
- obter leituras dos sensores do dispositivo.

O principal problema com esta divisão é determinar qual seria o formato dos dados que trafegam entre os módulos. Sabemos que a interface de comunicação deve ser fixa e independente de dispositivo; caso contrário, perde-se a independência entre os módulos.

Se estivermos trabalhando com um acelerômetro é natural que os dados provenientes do dispositivo sejam vetores tridimensionais (ou trios de variáveis numéricas, representando cada uma o movimento em uma das dimensões). Se em nossa interface estabelecermos que o módulo de dispositivo precisa gerar trios de variáveis numéricas para servir de entrada para o módulo de reconhecimento, estamos criando uma interface que só permite trabalhar com dispositivos que possuam acelerômetro; os dados de outros dispositivos não poderão ser representados nesse formato.

Se estivermos trabalhando com uma luva que possui medidores de flexão nos 5 dedos, os dados poderiam ser expressos em quíntuplas de variáveis numéricas, onde cada variável representaria a flexão de um dedo. Esta interface possibilitaria o uso de qualquer luva com medidores de flexão nos 5 dedos; porém, seria também inadequada para outros dispositivos, como uma luva com acelerômetro (que possui os dois tipos de medidores) ou mesmo para trabalhar com câmeras, cujas saídas são imagens.

É importante notar que, apesar desta possibilidade não permitir a independência total entre os módulos, ela ainda oferece versatilidade por permitir que não se saiba exatamente qual dispositivo está "do outro lado". Se estabelecermos uma interface para dispositivos com acelerômetro tridimensional, podemos teoricamente trabalhar com qualquer dispositivo desse tipo, já que todos usam o

mesmo formato de dados (ou podem ser facilmente adaptados para tal). O módulo de dispositivo agiria somente como um tradutor (driver) para o dispositivo poder se comunicar com o módulo de reconhecimento. De fato, os trabalhos correlatos ([E+03], [FSS98] e [HK93]) seguem essa metodologia. Apesar disso, esta possibilidade não nos dá o grau de independência que buscamos, o que nos levou a rejeitá-la.

3.1.2 Módulo de dispositivo responsável por obtenção e pré-processamento dos dados

Como visto na seção anterior, qualquer interface que utilize um formato de dados específico na comunicação entre os módulos seria capaz de funcionar apenas com dispositivos similares. Para obtermos versatilidade na escolha do dispositivo, precisamos de um formato de dados genérico. Consideremos agora a divisão de responsabilidades exposta na figura 3.3.

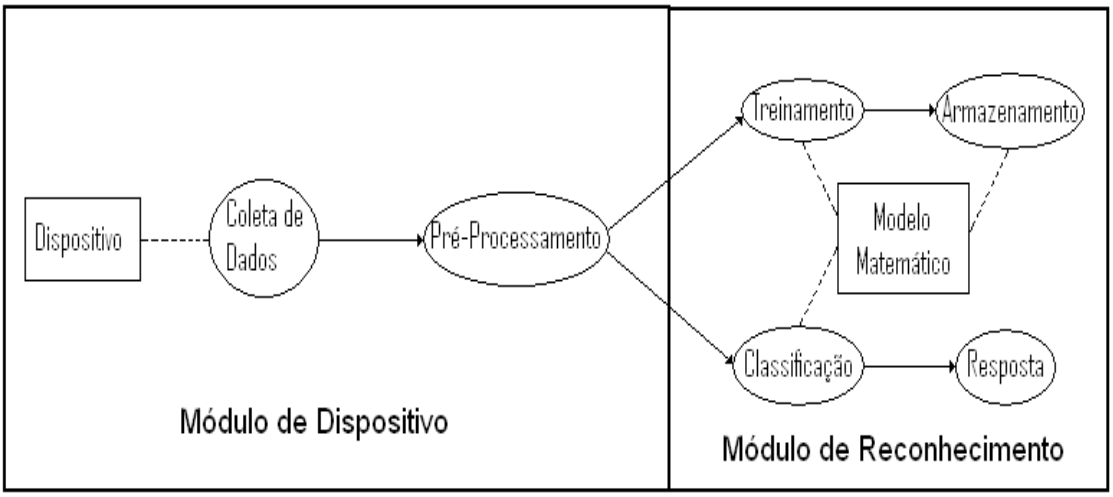


Figura 3.3: *Segunda divisão de responsabilidades entre os módulos.*

O módulo de dispositivo seria capaz de:

- iniciar ou interromper a comunicação com o dispositivo;
- obter leituras dos sensores do dispositivo, já filtradas e quantizadas.

Nesta divisão, o formato dos dados é determinado pela etapa de pré-processamento. Após a filtragem, os dados são submetidos ao processo de quantização, que associa um rótulo a cada leitura do dispositivo.

O modelo matemático usa os dados rotulados pelo quantizador para efetuar o treinamento e o reconhecimento, isto é, não são utilizados os dados puros nas duas etapas. Isso significa que

rotulação seria uma escolha natural de formato de dados, se houvesse uma rotulação que servisse para qualquer dispositivo. Mostraremos agora que tal rotulação genérica é possível se os rótulos forem considerados independentes.

3.1.3 Rotulações Específicas

Para ilustrar o problema, considere um dispositivo fictício D1 que é posicionado na mão direita do usuário e cujas leituras consistem na posição horizontal e vertical da mão em relação ao centro do corpo (no caso, o tórax), que é a origem do sistema. Quando o usuário move sua mão ao longo do plano, esse dispositivo emite constantemente a distância horizontal e vertical da mão ao tórax (a profundidade é ignorada). Considere também que o modelo matemático usado em conjunto com esse dispositivo necessita que seja feita a quantização dos dados emitidos para funcionar corretamente.

Um possível critério de quantização, que será usado neste exemplo, é a divisão do plano em 9 regiões. Cada região é rotulada de acordo com a sua posição em relação ao centro do corpo: *esquerda superior, acima, direita superior, esquerda, centro, direita, esquerda inferior, abaixo* e *direita inferior* (veja a figura 3.4). Cada leitura do dispositivo D1 é associada à região mais próxima, e o rótulo da região é a *observação* ocorrida naquele instante. O modelo matemático usado em conjunto com este dispositivo, portanto, espera receber como entrada sequências desses rótulos.

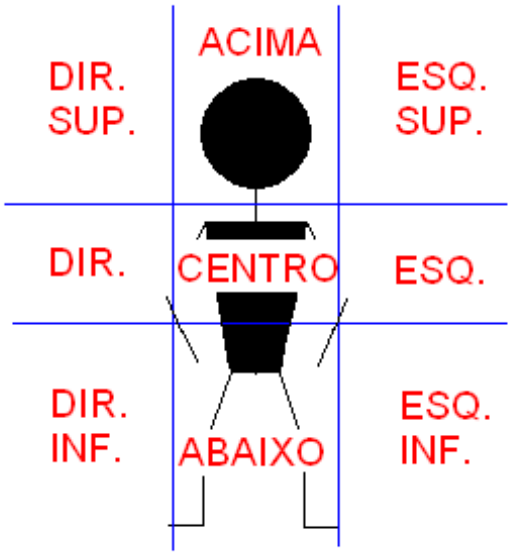


Figura 3.4: *Possível rotulação das regiões do plano.*

Suponha agora que este sistema seja treinado para reconhecer o gesto "para a esquerda", que consiste na seguinte sequência de observações: {"direita", "centro", "esquerda"}. Em seguida, o usuário efetua dois gestos, consistindo nas sequências {"direita", "centro", "esquerda superior"} e {"direita",

"centro", "direita superior"}, esperando que eles sejam classificados.

É evidente que os dois gestos efetuados não correspondem exatamente ao gesto treinado; eles diferem do gesto treinado por uma observação. Apesar disto, o sistema tenta classificá-los. Perceba que o modelo matemático poderia considerar o primeiro gesto mais provável de ser uma instância do gesto treinado do que o segundo, pelo fato de "esquerda superior" ser mais próximo de "esquerda" do que "direita superior". Neste caso, ele estaria usando *relações entre os rótulos* para fazer seu julgamento e consequentemente reconhecer com maior precisão.

Embora essa estratégia otimize o modelo matemático para trabalhar com aquele dispositivo em particular, ela impossibilita que usemos o mesmo modelo para trabalhar com outros dispositivos cujas relações entre rótulos sejam diferentes. Precisaríamos reprogramar o modelo levando em consideração as novas relações cada vez que trocássemos de dispositivo.

Considere um dispositivo com acelerômetro D2 que é posicionado na mão direita do usuário e a cada instante emite um vetor tridimensional que representa a aceleração do dispositivo relativa à posição de repouso. As leituras consistem de trios de variáveis numéricas, cada variável representando a intensidade da aceleração em um eixo (X, Y e Z).

Um possível critério de quantização para o dispositivo D2 é a divisão do espaço em 6 regiões. Cada região é rotulada de acordo com a sua posição em relação ao dispositivo: *acima, abaixo, frente, trás, esquerda e direita* (veja a figura 3.5). Cada região possui um vetor tridimensional associado a ela (os *codebook vectors*). Para determinar o rótulo de uma leitura do dispositivo, ela é transformada em um vetor tridimensional, e a distância desse vetor a cada um dos *codebook vectors* é calculada. A região do *codebook vectors* mais próximo ao vetor criado é escolhida como rótulo.

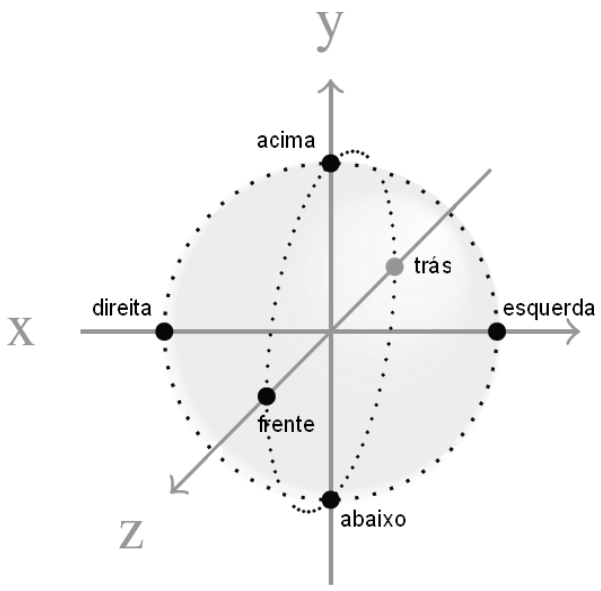


Figura 3.5: *Possível rotulação das regiões do espaço.*

Se desejássemos usar o dispositivo D2 em conjunto com o modelo matemático supracitado, já programado com as relações entre rótulos do dispositivo D1, seríamos obrigados a reprogramá-lo com as novas relações entre rótulos. Além disso, também é preciso re-treinar o sistema, pois gestos feitos com o dispositivo D1 são representados de outra forma (com outras observações) por D2.

Para obter a independência entre dispositivo e modelo precisamos criar uma *rotulação genérica*. Com uma rotulação genérica, os modelos matemáticos perdem alguma precisão por não saber as relações entre os rótulos, mas ainda são capazes de efetuar treinamento e reconhecimento.

3.1.4 Rotulações Genéricas

Suponha que no nosso exemplo, com o dispositivo D1, seja usado o mesmo critério de quantização, mas que sejam atribuídos os seguintes rótulos: *r1, r2, r3, r4, r5, r6, r7, r8, r9* (veja a figura 3.6). Ao treinarmos o gesto "para a esquerda", com esta rotulação sua sequência seria {"r6", "r5", "r4"}. Perceba que essa sequência é a única informação que o modelo matemático possuirá para decidir se outra sequência de observações é uma instância desse gesto ou não.

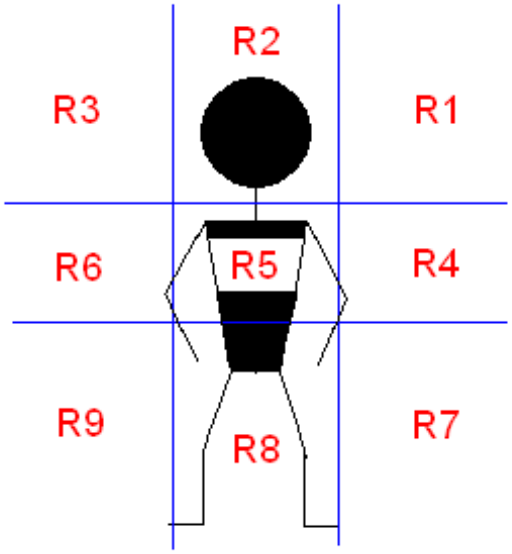


Figura 3.6: *Uma rotulação genérica das regiões do plano. Os rótulos não possuem significado associado a eles.*

Se o usuário efetuar os mesmos gestos, com a nova quantização serão geradas as sequências {"r6", "r5", "r1"} e {"r6", "r5" e "r3"}. Ao usarmos um modelo matemático genérico para tentar classificar os gestos, ele não tem como saber qual dos gestos é mais próximo do treinado porque não sabe as relações entre os rótulos. Isto é, "r4" é tão próximo de "r3" quanto de "r1". É importante que o modelo não tente inferir relações entre rótulos para não chegar a conclusões incorretas (no nosso exemplo, se ele usasse os números dos rótulos como critério de proximidade, ele concluiria

erroneamente que o segundo gesto é mais parecido com o treinado do que o primeiro).

Para compensar a perda das relações entre rótulos e permitir que o sistema do nosso exemplo avalie melhor o primeiro gesto são usados os conjuntos de treinamento. Se o primeiro gesto for uma variação comum do gesto treinado, espera-se que ele apareça ao menos uma vez no conjunto de treinamento, e o modelo matemático deve usar este fato para ter maior chance de classificá-lo corretamente. Desta forma, quanto maior o conjunto de treinamento, maior a chance de classificarmos corretamente pequenas variações do gesto treinado; grandes variações não devem aparecer no conjunto de treinamento (como o segundo gesto do exemplo), e portanto a chance de classificá-las como instância do gesto treinado é reduzida.

Usando uma quantização genérica com o dispositivo D2 da seção anterior, um possível resultado seriam os rótulos *r1, r2, r3, r4, r5 e r6* (veja a figura 3.7). Note que essa rotulação não tem nada em comum com a rotulação proposta de D1. Se desejássemos usar o dispositivo D2 em nosso sistema, a única coisa que seria preciso fazer é re-treinar o sistema para reconhecer gestos feitos com D2. Não seria necessário reprogramar o modelo matemático, pois ele não guarda informações sobre os rótulos, nem sequer o significado deles.

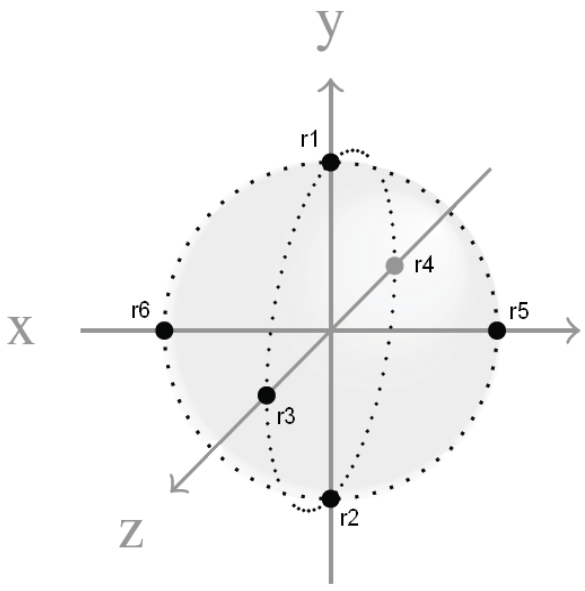


Figura 3.7: *Uma rotulação genérica das regiões do espaço.*

Treinar novamente o sistema pode parecer inconveniente, mas é necessário em muitas circunstâncias independente da arquitetura usada. Há gestos que podem ser feitos com um dispositivo e não com outros; no caso de troca de dispositivo, haveria a possibilidade de existir um gesto impossível de ser efetuado com o novo dispositivo. Além disso, se houver algum mecanismo de persistência de dados de treinamento, é possível treinar o sistema apenas uma vez para cada dispositivo, e

simplesmente restaurar os dados ao escolher o dispositivo correspondente.

Como visto, esta possibilidade traz algumas desvantagens, mas permite que qualquer dispositivo ou modelo matemático seja utilizado. O quantizador efetua um processo de abstração de dados; ele define características dos movimentos (as observações) baseado nas leituras dos dispositivos, e para cada gesto informa somente se cada característica ocorreu ou não, e quando ocorreu. O modelo matemático recebe somente as observações, não sabendo sobre as leituras do dispositivo que as originam; se ele não assumir relações entre as observações, podemos trocar o dispositivo e, treinando novamente o sistema, seremos capazes de efetuar o reconhecimento com o novo dispositivo sem maiores problemas. Devido a esse grau de desacoplamento entre dispositivo e modelo, esta possibilidade foi escolhida para o desenvolvimento do trabalho.

A figura 3.8 mostra o fluxo de informações entre os módulos sob esta divisão. Resumidamente, o módulo de dispositivo é um *produtor* de sequências de rótulos e o módulo de reconhecimento é um *consumidor* dessas sequências, sendo que o módulo de dispositivo produz os rótulos conforme o usuário efetua seus movimentos. As sequências podem ser usadas para dois fins: classificação do gesto, onde somente uma sequência é necessária, ou treinamento, onde múltiplas sequências, cada uma representando uma execução do gesto, são armazenadas.

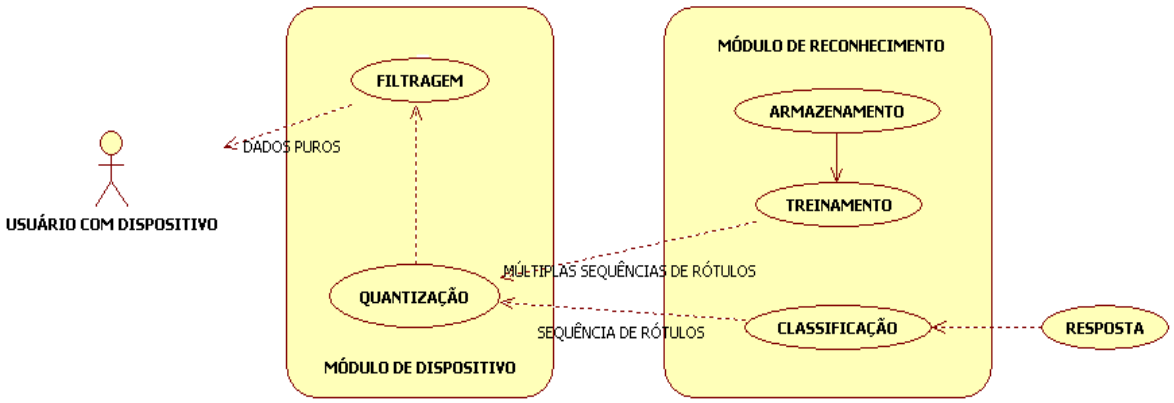


Figura 3.8: *Fluxo de informações sobre a segunda divisão proposta. As setas tracejadas indicam as dependências entre as etapas.*

Apesar dos módulos serem independentes, eles dependem totalmente do bom funcionamento dessa interface que define o formato de dados que trafegam entre os módulos. Então, ao criar um módulo de reconhecimento, surgem perguntas bastante pertinentes: o que o módulo deve esperar como entrada, e qual o comportamento esperado nos seus casos de uso? Como demonstrado na figura 3.9, na etapa de treinamento, serão associados conjuntos de sequências de rótulos a gestos, e o modelo matemático deve somente guardar de alguma forma esses dados; na etapa de classificação,

é apresentada uma sequência de rótulos ao modelo, e ele deve responder com o nome do gesto mais provável que corresponde àquela sequência, ou *null* se a sequência for muito diferente de todos os gestos treinados. É importante notar que a forma que o módulo faz as duas coisas depende somente da sua implementação; a interface proposta somente define o que deve ser feito pelo módulo. Outra observação relevante é que não necessariamente todo modelo matemático existente pode ter seu módulo criado para esta arquitetura; se o modelo não puder trabalhar com os conjuntos de rótulos da forma proposta, o módulo não será bom o bastante para ser usado na prática.

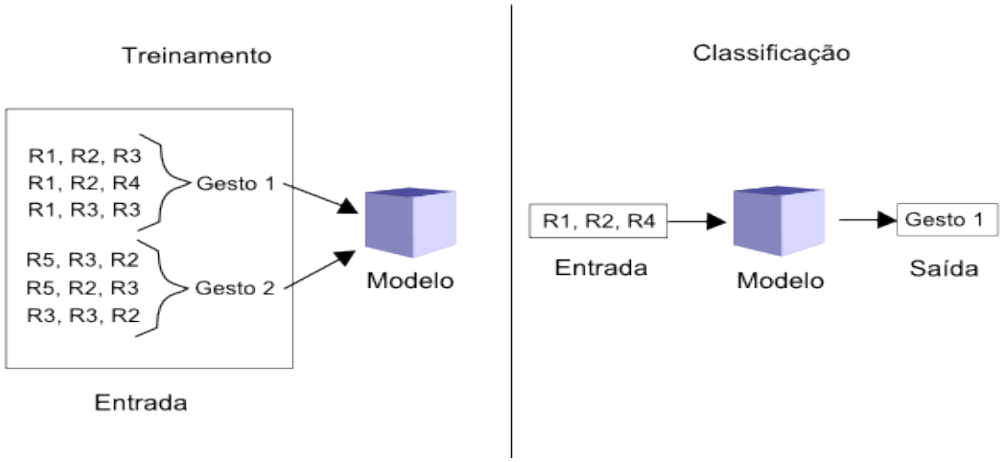


Figura 3.9: *Especificação do comportamento de um módulo de reconhecimento.*

Dúvidas similares aparecem na criação de um módulo de dispositivo. Como a arquitetura depende fortemente no mecanismo de quantização, o ponto principal é conseguir usar uma quantização adequada. Entretanto, não há uma maneira simples de determinar se uma quantização é boa o suficiente, porque a quantidade de informações pertinentes para o reconhecimento de gestos varia entre os dispositivos. Uma quantização escolhida pode ser simples demais e não diferenciar os possíveis estados do dispositivo o suficiente; uma outra pode ser excessivamente detalhada, exigindo um treinamento mais rigoroso do sistema para poder diferenciar sequências ligeiramente diferentes. Por isso, para criar um módulo de dispositivo é preciso testar diferentes mecanismos de quantização em conjunto com um módulo de reconhecimento confiável, até a obtenção de resultados satisfatórios. A figura 3.10 mostra o comportamento esperado de um módulo de dispositivo.

Uma consequência notável da abstração dos dados do dispositivo e do modo de operação dos modelos matemáticos é que meta-informações sobre a execução do gesto são perdidas. Características como posição, orientação, velocidade ou duração do gesto são normalmente inferidas a partir dos dados puros do dispositivo; entretanto, essas informações são usadas pelo módulo de dispositivo para gerar seus rótulos, e a partir do momento que os rótulos são transmitidos para o módulo de reconhecimento, as informações não são mais consideradas. Com o conjunto de rótulos em mãos, o

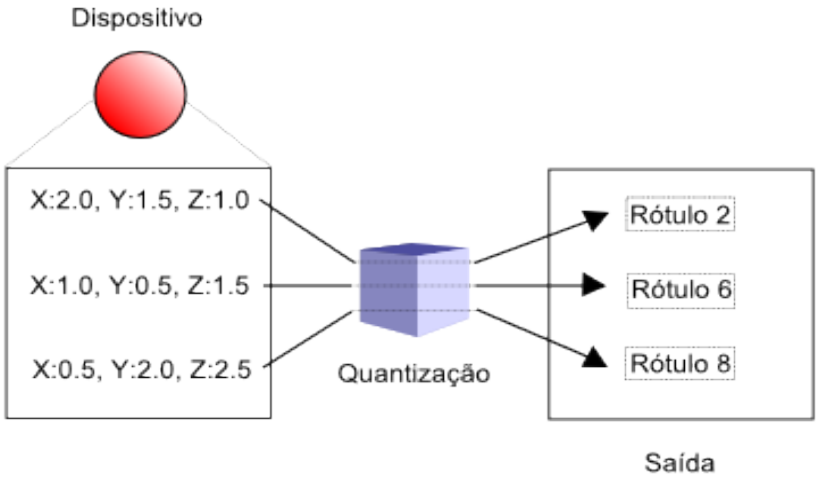


Figura 3.10: *Especificação do comportamento de um módulo de dispositivo.*

módulo de reconhecimento somente é capaz de dizer se um gesto ocorreu ou não, mas não *como* ele ocorreu. Por isso, esta arquitetura não é adequada para uso em contextos onde as meta-informações são importantes (por exemplo, navegação em mundos virtuais tridimensionais).

3.2 Refinamento das operações baseado em aspectos práticos

Após a definição da fronteira, é preciso levar em consideração alguns detalhes importantes do reconhecimento de gestos para aprimorar a interface. Nesta seção são apresentados esses detalhes e as mudanças que eles originaram nas responsabilidades dos módulos e na interface.

3.2.1 Obtenção de sequências de treinamento e reconhecimento

Um problema recorrente ao trabalhar com reconhecimento de gestos é determinar quando um gesto é iniciado ou terminado (também chamado *problema de segmentação*). Ao iniciar a comunicação com o dispositivo, o sistema deve possuir algum mecanismo que determine quando a comunicação deve ser interrompida e o conjunto de dados obtido deve ser classificado. Na divisão de responsabilidades proposta até agora não foi definido qual módulo se encarregaria dessa tarefa.

Considere a possibilidade de responsabilizar o módulo de reconhecimento pelo mecanismo. Desta forma ele interagirá com o módulo de dispositivo requisitando uma observação por vez, e repetirá esse procedimento até que decida que o conjunto de dados obtido é suficiente para classificar.

- Módulo de dispositivo (DISP)

Possui o algoritmo 1.

- Módulo de reconhecimento (REC)

  Possui o algoritmo 2.

Algoritmo 1: proxObs()

```
P ← dados puros do dispositivo;
O ← quantiza(P);
devolva O;
```

Algoritmo 2: Classifica()

```
C ← {};
enquanto C não é classificável faça
        O ← DISP.proxObs();
        Adicione O a C;
fim
devolva classificação de C;
```

Normalmente o módulo de reconhecimento não é capaz de determinar, dada uma única observação, se o gesto terminou ou não. Isso acontece porque em nossa arquitetura as observações são necessariamente desprovidas de semântica para atingir a independência de dispositivo e modelo. Entretanto, mesmo com essa limitação existem técnicas que podem ser usadas para possibilitar a classificação.

Uma dessas técnicas é o uso das chamadas janelas deslizantes (*sliding windows*), que são encontradas em inúmeros trabalhos (por exemplo, em sistemas de comunicação via rádio [WC01] e também reconhecimento contínuo de gestos [EKR98]). Ela consiste em guardar as J últimas observações ocorridas no movimento, e submetê-las para classificação. Se o modelo não consegue classificar um gesto com essa janela, a observação $J + 1$ é colocada na janela deslocando todas as outras para trás, e o processo se repete até o modelo conseguir classificar um gesto, ou terminarem as observações, ou chegar um limite de tentativas. O tamanho J da janela deve ser grande o suficiente para acomodar todos os gestos reconhecíveis pelo sistema, mas não pode ser excessivo para não atrapalhar o desempenho ou a acuidade da classificação.

Apesar da técnica apresentada permitir a classificação dos conjuntos de dados, ela não resolve o problema de treinamento do sistema. O treino consiste em múltiplas repetições de gestos que ainda não são reconhecíveis pelo sistema; se não houver alguma espécie de delimitador lógico entre as execuções, não é possível saber quando um gesto termina e o outro começa.

Para resolver os problemas de classificação e treinamento simultaneamente, considere agora a possibilidade de responsabilizar o módulo de dispositivo pelo mecanismo de determinação de início e fim de gesto. Desta forma, ao receber um pedido de classificação ou treinamento, o módulo de reconhecimento requisita observações do módulo de dispositivo até que este emita uma observação especial de fim de gesto; o módulo de reconhecimento pode então classificar o conjunto de dados obtido, se o pedido for de classificação, ou continuar obtendo outras execuções do mesmo gesto, se o pedido for de treinamento.

- Módulo de dispositivo (DISP)

  Continua somente com o algoritmo 1.

- Módulo de reconhecimento (REC)

  Passa a ter os algoritmos 3, 4 e 5.

Algoritmo 3: ObtemGesto()

```
C ← {};
O ← DISP.proxObs();
enquanto O ≠ fim-de-gesto faça
        |   Adicione O a C;
        |   O ← DISP.proxObs();
fim
devolva C
```

Algoritmo 4: Classifica()

```
C ← ObtemGesto();
devolva classificação de C
```

Algoritmo 5: Treina(numeroDeAmostras)

```
T ← {};
amostraAtual ← 1;
enquanto amostraAtual ≤ numeroDeAmostras faça
        |   C ← ObtemGesto();
        |   Adicione C a T;
        |   Incremente numeroDeAmostras;
fim
Armazene informações contidas em T;
```

Nesta abordagem o problema resume-se a decidir quando emitir a observação especial de fim de gesto. Cada módulo de dispositivo pode ter o seu critério, e há vários possíveis. Neste trabalho são usados dois critérios: a observação especial pode ser emitida após certo tempo de inatividade

Algoritmo 6: ObtemGesto()

```
C ← {};
O ← proxObs();
enquanto gesto não terminou faça
    |   Adicione O a C;
    |   O ← proxObs();
fim
devolva C
```

Algoritmo 7: ObtemSequenciaDeTreino(numeroDeAmostras)

```
T ← ∅;
amostraAtual ← 1;
enquanto amostraAtual ≤ numeroDeAmostras faça
    |   C ← ObtemGesto();
    |   Adicione C a T;
    |   Incremente numeroDeAmostras;
fim
devolva T
```

do dispositivo, ou com o pressionamento de um botão presente no dispositivo. Note que os dois critérios não dependem de interação com o modelo matemático.

A possibilidade discutida seria suficiente para resolver o problema, mas foi adotada uma solução que simplifica as interações entre os módulos: ao invés de obter observações individuais do módulo de dispositivo, pedimos *gestos* ou *sequências de treino*. No primeiro caso, o módulo de dispositivo já responde com uma sequência de observações, correspondente à execução de um gesto. No segundo caso o resultado é um conjunto de sequências de observações, que corresponde a todos os gestos feitos para treinar o modelo matemático. Ao fazer qualquer um dos pedidos, o módulo de reconhecimento simplesmente aguarda até que o módulo de dispositivo termine o seu trabalho, e opera sobre os dados resultantes. Isso implica que técnicas que dependam de interação com o modelo matemático para determinação de início ou fim de gestos tornam-se inviáveis com esta arquitetura (por exemplo, a técnica que usa as janelas deslizantes).

- Módulo de dispositivo (DISP)

  Passa a ter os algoritmos 1, 6 e 7.

- Módulo de reconhecimento (REC)

  Passa a ter os algoritmos 8 e 9.

Algoritmo 8: Classifica()

C ← DISP.ObtemGesto();
devolva *classificação de C*

Algoritmo 9: Treina(numeroDeAmostras)

T ← DISP.ObtemSequenciaDeTreino(numeroDeAmostras);
Armazene informações contidas em T;

3.2.2 Suporte a múltiplas observações por instante

Até agora consideramos somente a situação onde o módulo de dispositivo emite somente uma observação por instante. Como um exemplo, este é o caso de dispositivos cuja única informação relevante é um vetor tridimensional de aceleração; a cada instante, é aplicado o processo de quantização ao vetor lido, gerando um único rótulo.

Suponha que o processo de quantização desse módulo divide o espaço tridimensional em N partes; desta forma, há N rótulos diferentes que podem ser usados como saída do módulo. Não faria sentido neste caso haver mais de uma observação por instante, pois cada vetor é associado somente a um rótulo, e a presença de dois rótulos significaria dois vetores em um mesmo instante, o que é impossível. Perceba que neste caso os N rótulos são *mutuamente excludentes*, isto é, não podem aparecer simultaneamente em um mesmo instante como saída do módulo de dispositivo.

Alguns módulos de dispositivo podem ter mais de uma medida relevante ao reconhecimento de gestos. Como um exemplo, considere uma luva com medidores de flexão nos 5 dedos e acelerômetro. As flexões dos dedos são completamente independentes da aceleração da luva, portanto flexões podem ocorrer simultâneas aos vetores de aceleração. Para transmitir essas informações ao módulo de reconhecimento, o módulo de dispositivo é obrigado a usar mais de uma observação por instante; neste caso, uma representando a aceleração obtida, e cinco outras, presentes ou não, cada uma indicando a flexão de cada dedo.

O problema associado a esta situação é que alguns modelos matemáticos não são capazes de trabalhar com múltiplas observações por instante. Um exemplo são os Modelos Ocultos de Markov tradicionais, onde no conjunto de observações $O = \{o_1, o_2, ..., o_t\}$, cada o_i corresponde a um número inteiro. Ao receber dados de um módulo que emite múltiplas observações por instante tal modelo matemático certamente ignoraria as observações excedentes, o que eventualmente levaria a inconsistências de treinamento e classificação. Ao mesmo tempo não é desejável proibir que sejam criados módulos de reconhecimento que usem tais modelos. Por isso é necessário algum mecanismo que permita descobrir a compatibilidade entre os módulos.

A solução proposta consiste em adicionar à interface de comunicação entre os módulos uma operação que indique se o módulo em questão oferece ou não suporte a múltiplas observações por instante. Assim, ao escolher um módulo de dispositivo, pode-se invocar esta operação para descobrir se o módulo oferece suporte. Se sim, o módulo só funcionará corretamente com módulos de reconhecimento que também ofereçam suporte; caso contrário, não há restrição sobre o módulo de reconhecimento. É importante que o módulo forneça a resposta correta de acordo com seu comportamento, ou os resultados não são definidos (alguns modelos matemáticos podem causar erros de execução, outros podem simplesmente ignorar a informação extra e fornecer resultados completamente incorretos).

3.3 Implementação

Para a implementação da interface, da biblioteca de reconhecimento de gestos e dos módulos, foi escolhida a linguagem de programação Java, por causa da familiaridade do autor com a linguagem e pelo fato da biblioteca que originou dois dos módulos criados (a biblioteca Wiigee [SPHB08]) ter sido feita em Java.

3.3.1 Definições

Deste ponto em diante serão utilizadas as seguintes definições:

- O conjunto $\Psi = \{\psi_1, \psi_2, ..., \psi_m\}$ representa todas as possíveis observações emitíveis pelo dispositivo sendo usado.
- Os elementos ι_x, que contém um ou mais elementos do conjunto Ψ, representam as observações emitidas pelo dispositivo no instante x.
- O conjunto $E = \{\iota_1, \iota_2, ..., \iota_t\}$ representa uma execução de gesto, sendo t a duração total do gesto e cada um dos elementos ι_x como definidos acima.
- O conjunto $\Sigma = \{\epsilon_1, \epsilon_2, ..., \epsilon_w\}$ representa o conjunto de treinamento para um gesto, sendo cada um dos elementos ϵ_x uma execução de gesto como definido acima.
- O conjunto $\Gamma = \{\sigma_1, \sigma_2, ..., \sigma_k\}$ representa todos os gestos que desejamos reconhecer, sendo cada um dos elementos σ_x um conjunto de treinamento como definido acima.

3.3.2 Transformação dos conceitos presentes na interface em tipos Java

Existem inúmeras formas de implementar a interface estudada em Java. Aqui serão expostas as escolhas de tipos Java que foram feitas para representar cada conceito.

- *Classificações de gestos*: para o computador, pouco importa a forma em que são representadas as classificações de gestos. Uma abordagem natural é rotular numericamente cada gesto na ordem que são treinados; assim, o gesto número 1 é o primeiro gesto treinado, o número 2 é o segundo, e assim por diante. Entretanto, para o usuário é melhor que gestos sejam associados a nomes, porque nomes são mais fáceis de lembrar do que a ordem em que foram treinados os gestos. Assim o usuário pode nomear um gesto circular como "círculo" ao invés de reconhecê-lo por um número. Por esse motivo, foi escolhido o tipo *String* de Java, que representa uma cadeia de caracteres. Portanto, a cada σ_x é associada uma palavra, a descrição do gesto.

- *Observações*: cada observação é uma leitura de algum medidor do dispositivo depois de submetida ao processo de quantização. Observações são as unidades básicas de comunicação entre os módulos de dispositivo e modelo, e podem ser representadas de qualquer forma, desde que não haja ambigüidade. Foi decidido rotulá-las com números inteiros, representados pela classe *Integer* de Java. Portanto, cada ψ_x será representado por um valor entre 0 e $m-1$. É importante relembrar que a interface assume que os rótulos são independentes, ou seja, uma observação de valor 5 é tão diferente de uma de valor 6 quanto de uma de valor 10. Assim a magnitude dos números deve ser desconsiderada pelo módulo de reconhecimento.

- *Intervalos*: um intervalo é caracterizado como um conjunto de observações ocorridas em um instante (os elementos ι_x definidos anteriormente). Qualquer estrutura que represente uma coleção seria adequada para agrupar observações, visto que ordem não é necessária dentro do mesmo intervalo. Foram escolhidas as listas de Java para facilitar a implementação; assim, um intervalo (ou *Interval*) possui uma *List* de Integers, ou observações.

- *Execuções de Gestos*: uma execução de gesto consiste de uma coleção ordenada de intervalos, onde cada intervalo guarda as observações obtidas naquele instante. Como a ordem dos intervalos é essencial, as listas de Java foram escolhidas. Uma execução de gesto é representada pela classe *Gesture*, cujas instâncias possuem uma List de Intervals. Uma execução de gesto corresponde a um elemento ϵ_x como definido anteriormente.

- *Conjuntos de Treinamento*: cada conjunto de treinamento é composto de uma coleção de execuções de gestos, que são usadas junto com um algoritmo para treinar o modelo matemá-

tico. Embora a ordem não seja um requisito, foram escolhidas novamente as listas de Java por conveniência. Portanto, uma sequência de treino (ou *TrainSequence*) possui uma List de Gestures. Elas correspondem aos elementos σ_x como definidos acima.

3.3.3 Interfaces de dispositivo e modelo matemático

Com os tipos definidos, torna-se possível a especificação dos métodos que poderão ser invocados em cada módulo. Os métodos foram agrupados em Interfaces Java, e classes que desejem se comportar como módulos devem simplesmente implementar os métodos da interface.

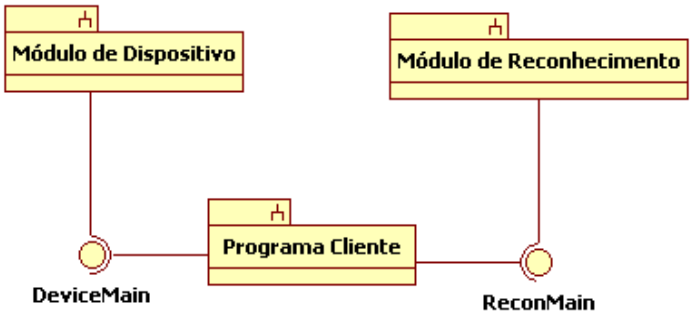


Figura 3.11: *Os módulos, as interfaces por eles implementadas e sua relação com o programa cliente.*

A figura 3.11 mostra a relação entre os módulos e o programa cliente, no qual se deseja usar o reconhecimento de gestos. Para isso, ele comunica-se com o módulo de dispositivo através da interface *DeviceMain*, e com o módulo de reconhecimento através da interface *ReconMain*. O próprio programa cliente fica com a obrigação de coordenar os dois componentes e passar os dados fornecidos pelo módulo de dispositivo para o módulo de reconhecimento.

- DeviceMain, interface para módulos de dispositivo

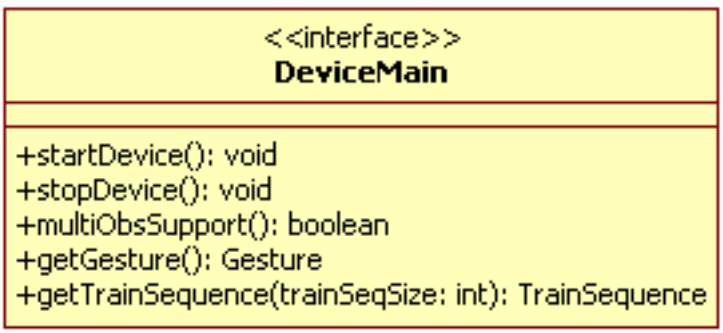


Figura 3.12: *A interface DeviceMain.*

startDevice(): efetua os preparos necessários para que o dispositivo funcione adequadamente. Após a chamada deste método, o módulo de dispositivo deve ser capaz de efetuar chamadas aos métodos getGesture() e getTrainSequence().

stopDevice(): efetua os preparos para desativar o dispositivo.

multiObsSupport(): devolve uma variável booleana que indica se o módulo em questão trabalha com múltiplas observações por instante, isto é, se as listas de observações dos seus intervalos podem ter tamanho maior que 1.

getGesture(): obtém uma execução de gesto feita com o dispositivo e aplica os processos de filtragem e quantização, obtendo uma instância de Gesture, que será analisada pelo módulo de reconhecimento.

getTrainSequence(trainSeqSize): obtém o número de execuções de gesto determinado pelo parâmetro trainSeqSize, aplica os processos de filtragem e quantização, obtendo várias instâncias de Gesture, e as coloca em uma instância de TrainSequence, que será usada pelo módulo de reconhecimento no treinamento.

- ReconMain, interface para módulos de reconhecimento

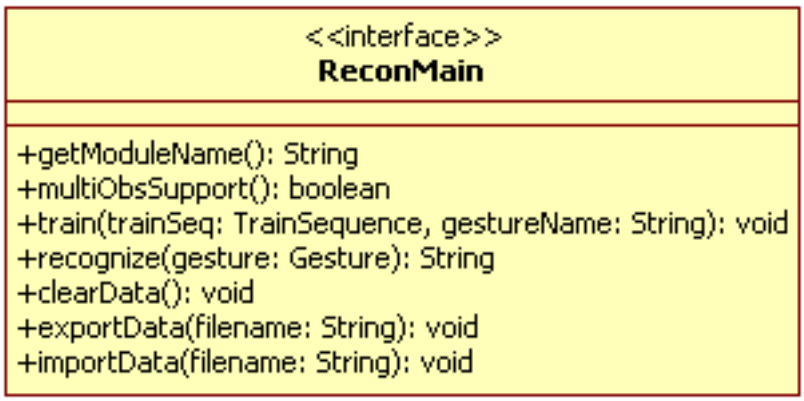


Figura 3.13: *A interface ReconMain.*

getModuleName(): devolve o nome deste módulo. É usado pelo mediador para criar a pasta onde são armazenados os dados salvos de reconhecimento.

multiObsSupport(): devolve uma variável booleana que indica se o módulo em questão trabalha com múltiplas observações por instante, isto é, se ele é capaz de processar listas de observações de intervalos que possuam tamanho maior que 1.

train(trainSeq, gestureName): recebe um TrainSequence e utiliza seus dados para treinar o modelo matemático. O parâmetro gestureName é usado como a classificação do gesto, e é a resposta dada ao tentarmos reconhecer o gesto com o método recognize(). Ao tentar treinar outro gesto com o mesmo nome, os dados do gesto antigo devem ser substituídos.

recognize(gesture): recebe um Gesture e tenta descobrir qual dos gestos treinados é mais semelhante ao fornecido, respondendo com o nome do gesto mais parecido, ou com *null* se nenhum dos gestos treinados é similar o suficiente.

clearData(): apaga todos os dados de treinamento. Após este método, o modelo é incapaz de reconhecer qualquer gesto.

exportData(filename): salva os dados de treinamento em um arquivo de nome filename.

importData(filename): carrega dados de treinamento a partir do arquivo de nome filename, criado com o método exportData().

- Descrição dos métodos e da interface do mediador

As interfaces apresentadas já fornecem a funcionalidade necessária para efetuar o reconhecimento, mas é importante padronizar as interações entre os módulos para tornar a biblioteca mais fácil de usar; em especial, para retirar a responsabilidade de coordenação entre os componentes do programa cliente. Para isso foi criado o mediador, cujo objetivo é abstrair as comunicações entre os módulos e oferecer somente operações mais simples para o programa cliente. O mediador é um exemplo característico de aplicação do padrão de projeto conhecido como Mediator [GHJV95].

A inclusão do mediador permite que a coordenação entre os componentes seja abstraída, e somente operações mais simples sejam diretamente requisitadas ao mediador, como escolha de módulos, inicialização ou parada de dispositivo, reconhecimento de um gesto, treinamento de um gesto, e operações sobre os dados de treinamento armazenados (salvar, carregar ou apagar).

setDeviceModule(moduleName): abre um arquivo JAR que contém um módulo de dispositivo e passa a utilizá-lo. Usa os métodos multiObsSupport() dos dois módulos para garantir que são compatíveis.

setReconModule(moduleName): faz o mesmo para o módulo de reconhecimento.

startDevice(): inicializa o dispositivo do módulo escolhido, usando seu método startDevice().

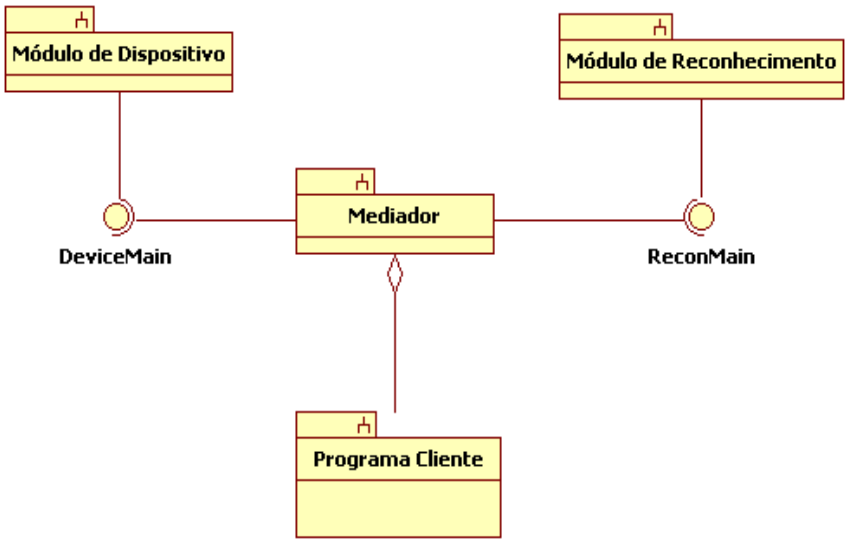


Figura 3.14: *Os relacionamentos entre as classes após a inclusão do mediador.*

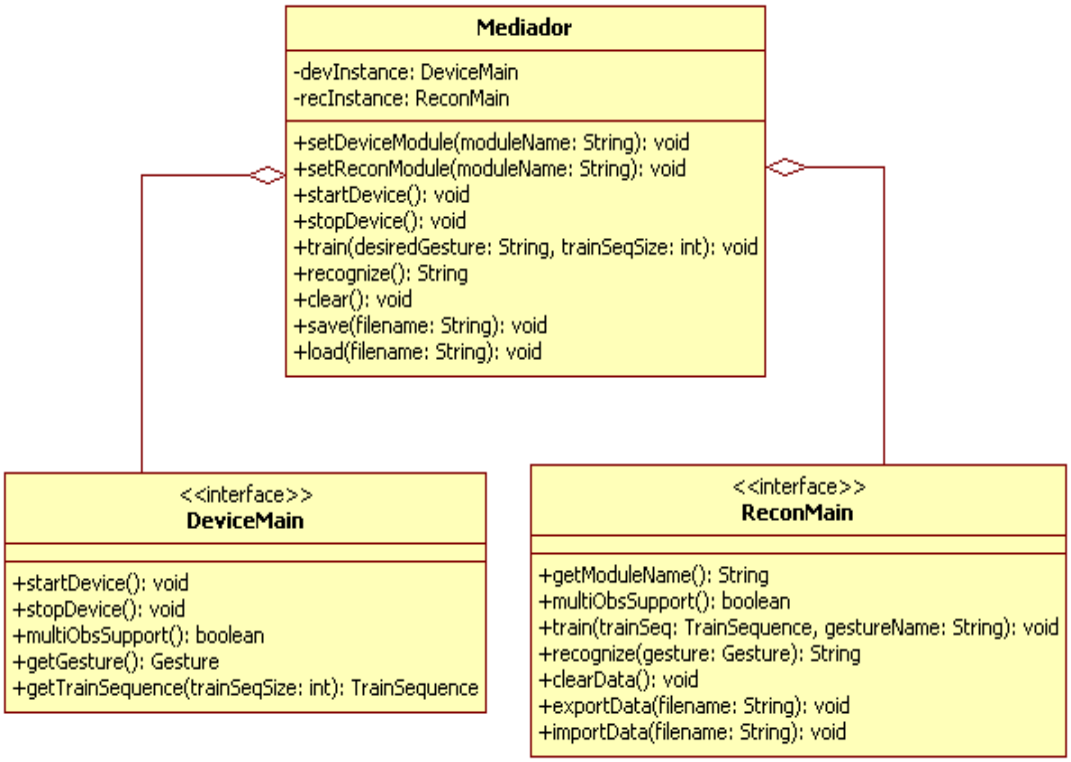


Figura 3.15: *Diagrama de classes com os métodos do mediador.*

stopDevice(): desativa o dispositivo do módulo escolhido, usando seu método stopDevice().

train(desiredGesture, trainSeqSize): treina o gesto de nome definido pelo parâmetro desiredGesture, pelo número de vezes definido pelo parâmetro trainSeqSize. Ele invoca o método getTrainSequence() do módulo de dispositivo, passando o resultado para o método train() do módulo de reconhecimento.

recognize(): pega um gesto feito com o dispositivo e tenta classificá-lo com o modelo matemático, devolvendo o nome do gesto mais parecido (ou *null*). Ele invoca o método getGesture() do módulo de dispositivo, passando o resultado para o método recognize() do módulo de reconhecimento.

clear(): apaga todos os dados de treinamento, invocando o método clear() do módulo de reconhecimento.

save(filename): salva os dados de treinamento em um arquivo de nome filename, invocando o método exportData() do módulo de reconhecimento.

load(filename): carrega os dados de treinamento de um arquivo de nome filename, invocando o método importData() do módulo de reconhecimento.

Perceba que um usuário do Mediador não precisa nem mesmo conhecer os tipos Interval, Gesture e TrainSequence, que são usados internamente.

3.3.4 GRMediator: detalhes do funcionamento da biblioteca

A premissa básica da biblioteca de reconhecimento proposta é ser capaz de trabalhar com módulos. Isso significa que sua implementação não deve fazer referência a nenhum módulo em particular, tanto durante a compilação quanto a execução. Os módulos devem ser fornecidos em tempo de execução, e a biblioteca deve ser capaz de carregar o módulo dado e instanciar um objeto da classe que implementa uma das interfaces para então poder invocar os métodos. Foi utilizado o mecanismo de *reflexão* (reflection) de Java para possibilitar esse comportamento.

O mecanismo de reflexão de Java permite que um programa carregue classes durante sua execução. Para isso, ele precisa saber o nome da classe e a qual pacote ela pertence. Procurar por uma classe que implemente uma das interfaces em um módulo exigiria que fossem averiguados todos os pacotes daquele módulo, e ainda haveria a possibilidade de escolher a classe errada, se houvesse mais de uma classe que de fato implementasse alguma das interfaces. Para eliminar esses problemas, um padrão foi estabelecido: a biblioteca recebe um arquivo JAR como entrada, e procura dentro

desse arquivo a classe *Main* dentro do pacote *devmain*, se for um módulo de dispositivo, ou do pacote *recmain*, se for um módulo de reconhecimento. A biblioteca espera que seja essa a classe que implementa a interface, e efetua verificações para garantir essa condição, recusando o módulo se a condição não for satisfeita. Também é esperado que o módulo contenha todas as classes de que precisa para funcionar corretamente.

Módulos de dispositivo devem ser colocados na pasta *devices*, e módulos de reconhecimento na pasta *models*. Dados de reconhecimento são salvos e carregados dentro de uma pasta com o nome do módulo de reconhecimento atual, que é criada dentro da pasta *data*, que por sua vez é criada dentro da pasta *models*. Esses comportamentos podem ser alterados no código fonte.

O pacote *interfaceClasses* possui todas as classes que devem ser usadas na criação de novos módulos. Basta criar um arquivo JAR com esse pacote e adicioná-lo às dependências do novo módulo. Quando for criado o JAR do novo módulo, não é necessário incluir essas classes, pois elas já estão presentes na biblioteca.

A classe que serve de interface com o programa cliente é a classe *Mediator* do pacote main da biblioteca. Uma instância dessa classe é um sistema de reconhecimento de gestos completo; vários sistemas podem ser criados para reconhecer gestos de múltiplos dispositivos de entrada, uma característica interessante para jogos.

Aqui temos um exemplo de uso do mediador em um programa cliente chamado TesteMediador:

```
1 public class TesteMediador {
2   public static void main(String[] args) {
3     Mediator m = new Mediator();
4     try {
5       // escolhe o wiimote como dispositivo
6       m.setDeviceModule("WDM.jar");
7       // escolhe o HMM como modelo
8       m.setReconModule("HMMRM.jar");
9     } catch (InadequateModuleException e) {
10    // isto acontecerá se algum dos módulos tiver algum problema
11      System.out.println("Módulo inadequado, encerrando programa...");
12      System.exit(0);
13    }
14    try {
15      // inicia o dispositivo
16      m.startDevice();
17      // treina o gesto g1 20 vezes
18      m.train("gesto1", 20);
```

```
19          // treina o gesto g2 20 vezes
20          m.train("gesto2", 20);
21          for (int i = 0; i < 10; i++) {
22            // reconhece 10 vezes o gesto feito pelo usuário
23            System.out.println(m.recognize());
24          }
25
26        } catch (IOException e) {
27        // isto acontecerá se houver falha de comunicação com o dispositivo
28          e.printStackTrace();
29        } finally {
30          try {
31          // pára o dispositivo
32            m.stopDevice();
33          } catch (IOException e) {
34          // isto acontecerá se houver algum problema na tentativa de parar o
                dispositivo
35            e.printStackTrace();
36          }
37        }
38    }
39 }
```

Na primeira linha do método main um mediador é instanciado. Em seguida é usado o método setDeviceModule para escolher o módulo de dispositivo. Neste exemplo, é buscado o arquivo WDM.jar dentro da pasta devices. Se o módulo não existir ou se sua estrutura não estiver no formato esperado, ocorre uma exceção. É feito o mesmo para o módulo de reconhecimento com o método setReconModule, buscando o arquivo HMMRM.jar dentro da pasta models.

Com os módulos determinados, o programa tenta iniciar o dispositivo escolhido com o método startDevice. Cada dispositivo tem um processo diferente; caso ocorra alguma falha nessa etapa, uma exceção é lançada. Após essa chamada o dispositivo está pronto para ser usado. Neste programa são treinados dois gestos, chamados de "gesto1" e "gesto2". Ao executar a primeira chamada, o programa cliente aguarda até que o processo de treinamento do "gesto1" tenha terminado. Nesse ponto o usuário fará 20 repetições do gesto que deseja treinar; essas informações são transmitidas e armazenadas no módulo de reconhecimento (no HMM neste exemplo). O mesmo é feito para o "gesto2".

Após o treinamento o programa está pronto para reconhecer gestos. Neste exemplo ele tenta

reconhecer 10 gestos feitos pelo usuário. A chamada a recognize faz o programa cliente aguardar até que o gesto tenha sido efetuado. Depois que o usuário faz o movimento os dados são transmitidos ao módulo de reconhecimento e este efetua a classificação, devolvendo a String com o nome do gesto reconhecido ou *null* se nenhum gesto for reconhecido. Neste exemplo o nome do gesto só é usado para imprimir na tela. Finalmente, o dispositivo é interrompido com o método stopDevice.

É importante notar que um programa escrito desta forma obriga que os gestos sejam re-treinados toda vez que ele for iniciado. Com a capacidade de salvar e carregar dados, treinamentos já feitos podem ser aproveitados. Com o programa cliente em desenvolvimento, são criados os arquivos que contém os dados de treinamento:

```
1 try {
2   // inicia o dispositivo
3   m.startDevice();
4   // treina o gesto g1 20 vezes
5   m.train("gesto1", 20);
6   // treina o gesto g2 20 vezes
7   m.train("gesto2", 20);
8
9   ... // quaisquer outros treinos
10
11  // salva os dados de treinamento em um arquivo
12  m.save("gestos.data");
13 }
14 ...
```

O seguinte código carrega os dados de treinamento do arquivo e já permite que os gestos treinados sejam reconhecidos:

```
1 try {
2   // inicia o dispositivo
3   m.startDevice();
4   // carrega os dados de treinamento
5   m.load("gestos.data");
6 for (int i = 0; i < 10; i++) {
7   // reconhece 10 vezes o gesto feito pelo usuário
8     System.out.println(m.recognize());
9   }
10 }
11 ...
```

Isto permite que todo o treinamento seja feito pelos desenvolvedores, livrando os usuários da tarefa desnecessária. Deve-se exercer cuidado, entretanto, pois os dados salvos são criados pelo módulo de reconhecimento em uso, e normalmente esses dados só fazem sentido se for usado o mesmo módulo de dispositivo que os enviou.

Com uma instância do mediador adequadamente treinada, podemos esboçar um código que fica analisando os gestos feitos pelo usuário indefinidamente e ativando funcionalidades do programa de acordo com o gesto efetuado:

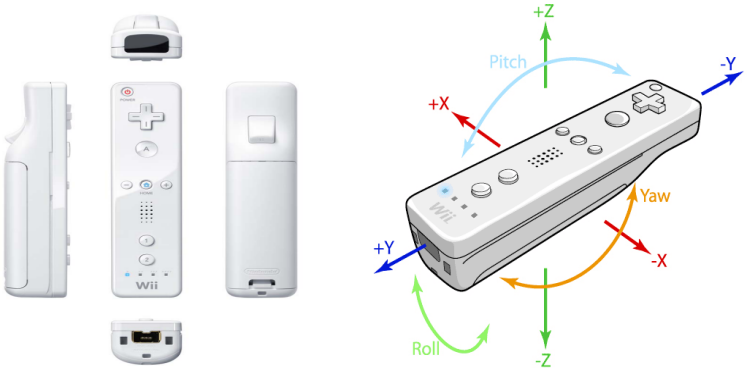
```
1 try {
2   // inicia o dispositivo
3   m.startDevice();
4   while (true) {
5     String gesto = m.recognize();
6     if (gesto == null) {
7       ... // nenhum gesto foi reconhecido
8     }
9     else if (gesto.equals("gesto1")) {
10      ... // executa a ação associada ao gesto 1
11    }
12    else if (gesto.equals("gesto2")) {
13      ... // executa a ação associada ao gesto 2
14    }
15    ...
16  }
17 }
18 ...
```

3.3.5 WiimoteDeviceModule: módulo de dispositivo que trabalha com o Nintendo Wiimote

O WiimoteDeviceModule (ou WDM) foi o primeiro módulo de dispositivo criado neste trabalho. Foi construído por modificações no código da biblioteca Wiigee [SPHB08], que originou também o módulo de reconhecimento de Modelos Ocultos de Markov.

O Nintendo Wiimote é bastante similar a um controle remoto. Além de vários botões digitais, possui também um acelerômetro de três eixos, e os dados de aceleração são transmitidos via Bluetooth [wii]. Essas informações são normalmente recebidas e processadas pelo Nintendo Wii, mas já estão disponíveis vários programas que permitem que os dados do Wiimote sejam recebi-

dos e interpretados por computadores convencionais, bastando que estes possuam uma interface de comunicação Bluetooth adequada.



(a) O Nintendo Wiimote. (b) Os eixos de movimento que o controle utiliza.

Figura 3.16: *O Nintendo Wiimote*

Ao invocar o método startDevice, o módulo usará a interface Bluetooth para procurar por Wiimotes nos arredores. O Wiimote a ser usado deve ter seus botões 1 e 2 pressionados para ser detectado pelo módulo. Desse ponto em diante os métodos que dependem de comunicação com o dispositivo (getGesture ou getTrainSequence) podem ser invocados. O módulo foi construído de forma a trabalhar com apenas um Wiimote por vez. Ao término do programa é importante que o método stopDevice seja invocado para encerrar a comunicação Bluetooth com o Wiimote.

Os dados emitidos pelo acelerômetro são as únicas leituras de interesse para o reconhecimento de gestos. Ao receber um pedido de dados (via getGesture ou getTrainSequence) o módulo aguarda até que o usuário segure o botão B do Wiimote. Isso é interpretado como um sinal de início de gesto, o que inicia a coleta de dados de aceleração. Dois processos de filtragem são efetuados: os dados são armazenados somente se a aceleração lida não é considerada desprezível (com magnitude abaixo de determinado valor) e se ela for consideravelmente diferente da última aceleração lida (o filtro de equivalência direcional [SPHB08]). Esse procedimento é repetido até que o usuário solte o botão B do Wiimote, o que é interpretado como um sinal de fim de gesto.

Se o sistema está em treinamento, esse processo é repetido de acordo com o número de vezes passado como parâmetro no método getTrainSequence. Depois de terminado é usado um quantizador padrão para transformar os dados de treinamento no formato esperado pela interface. O quantizador divide o espaço tridimensional em 14 partes usando o *algoritmo das k médias* (veja [SPHB08] para detalhes), mapeando cada vetor tridimensional a um dos 14 valores. Assim, os intervalos gerados por este módulo constituem-se sempre de uma única observação, um valor de 0 a 13, e o módulo não trabalha com múltiplas observações por instante.

Se o sistema está tentando reconhecer um gesto, o conjunto de acelerações obtidas é passado para o quantizador padrão, que gera um objeto da classe Gesture. Esse objeto é então enviado para o módulo de reconhecimento, que efetuará a classificação.

3.3.6 HiddenMarkovModelsRecognitionModule: módulo de reconhecimento que usa Modelos de Markov Ocultos

O HiddenMarkovModelsRecognitionModule (ou HMMRM) foi o primeiro módulo de reconhecimento criado neste trabalho. Foi também construído por modificações no código da biblioteca Wiigee [SPHB08].

A cada vez que um gesto novo é treinado (via train), um GestureModel é criado. Cada GestureModel representa o modelo de um gesto reconhecível pelo sistema. Instâncias dessa classe mantêm uma referência a um Modelo Oculto de Markov (ou HMM), e aplicam o algoritmo de treinamento (um *algoritmo de Baum-Welch* [Rab89] modificado) sobre ele. A classe GestureManager guarda as associações entre nomes de gestos e os GestureModels correspondentes.

O número de estados N de todos os HMMs é fixo (igual a 8) e foi determinado empiricamente [SPHB08]. Já a matriz B (probabilidade de emissão de uma observação dado um estado) precisa ter dimensões N x M, onde M é o número total de observações diferentes (ou tamanho do conjunto V [2.1.2]). Na implementação de nossa arquitetura em Java, cada rótulo é um número inteiro, o que permite uma correspondência direta entre os rótulos e os valores do conjunto V, ou seja, $v_i = \psi_i, 1 \leq i \leq M$ [3.3.1]. Como M varia de dispositivo para dispositivo, ele é estimado a partir do conjunto de treinamento, e tem o valor da maior observação que nele ocorre:

$$M = max_{1 \leq i \leq |\Psi|}\{\psi_i\}, \text{ para todo } \iota_x. \tag{3.1}$$

Caso ocorra uma observação maior que M no momento do reconhecimento, ela é ignorada dentro do GestureModel.

Para efetuar o reconhecimento de um gesto (com recognize), o módulo passa o objeto Gesture recebido para cada um dos GestureModels, que calculam a probabilidade daquele movimento ser de fato uma execução do gesto representado por si (usando o *forward algorithm* [Rab89]). Após todas as probabilidades serem calculadas, um *classificador Bayesiano* é aplicado para decidir qual dos gestos será dado como resposta. Se nenhum deles for considerado provável o suficiente, o método devolve *null*.

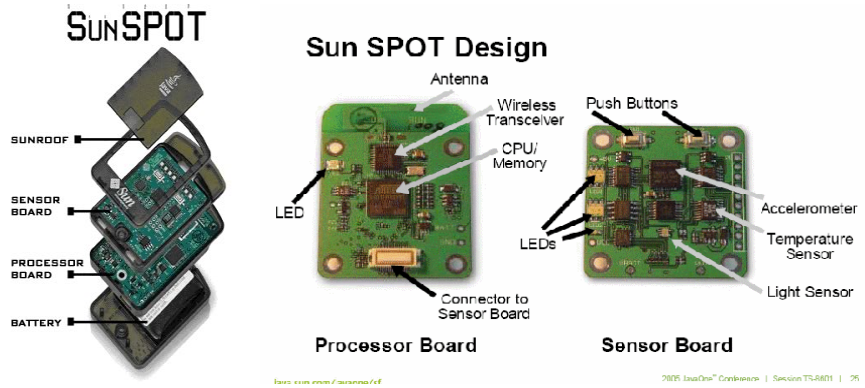
Os HMMs usados neste módulo assumem que ocorre somente uma observação por instante

($|\iota_x| = 1$), portanto ele não é capaz de trabalhar em conjunto com módulos de dispositivo que emitam mais de uma observação por instante. Adaptações nos modelos para que aceitem múltiplas observações por instante podem ser feitas (como mencionado em [Män01]) e são um próximo passo natural da implementação deste módulo.

3.3.7 SunSPOTDeviceModule: módulo de dispositivo que trabalha com os Sun SPOTs

O SunSPOTDeviceModule (ou SPOTDM) foi o segundo módulo de dispositivo criado neste trabalho. O objetivo desse módulo é permitir o uso dos Sun SPOTs como alternativa de dispositivo.

Um dispositivo Sun SPOT (Small Programmable Object Technology) é uma plataforma experimental desenvolvida quase totalmente em Java [Sun]. Um Sun SPOT livre (free-range) consiste de três partes: uma placa com um microprocessador e rádio, uma camada com baterias e uma placa com sensores de luz, temperatura, aceleração, pinos de entrada e saída, 2 botões e 8 LED's (Lights Up Displays). Um Sun SPOT estação-base (basestation) consiste apenas da placa com microprocessador e rádio. Essa estrutura permite que sejam usados SPOTs livres para capturar informações do ambiente, realizar processamentos em programas simples e trocar informações entre SPOTs via rádio.



(a) As camadas que compõem um Sun SPOT.

(b) Componentes internos do SPOT.

Figura 3.17: *O Sun SPOT*

Dos sensores com os quais os SPOTs são equipados, somente o acelerômetro tem aplicação no reconhecimento de gestos. Ao contrário do Nintendo Wiimote, os SPOTs não ficam constantemente enviando dados. É necessário que seja criado um programa, instalado no SPOT, que determine quais dados serão enviados, e como. Deve também haver outro programa rodando em um computador que recebe os dados e transforme-os no formato esperado pela interface de módulo de dispositivo. Assim,

o SPOTDM consiste de dois programas: um que deve estar instalado no SPOT do qual desejamos obter dados, e outro que é escolhido pela biblioteca como módulo de dispositivo e roda em um computador. O primeiro é uma adaptação de um demonstrativo de código aberto da tecnologia, fornecido no próprio kit de desenvolvimento (aplicativo de *Telemetria*). O segundo utiliza uma estação-base conectada ao computador para receber dados via rádio.

Ao invocar o método startDevice, o módulo inicia uma transmissão de rádio buscando por SPOTs livres nas imediações. SPOTs que tiverem o programa de telemetria instalado e rodando respondem a essa transmissão, estabelecendo conexão com o módulo e iniciando um processo de calibração, durante o qual o SPOT deve permanecer imóvel. Com o fim do processo é possível fazer chamadas a getGesture ou getTrainSequence. O módulo permanece recebendo dados do mesmo SPOT até que o método stopDevice seja chamado, o que encerra a comunicação e coloca o SPOT em espera por uma nova transmissão.

Ao receber um pedido de dados (via getGesture ou getTrainSequence) o SPOT começa a analisar os dados de aceleração. Se de um instante ao próximo houver uma variação de magnitude grande o suficiente no vetor de aceleração, um gesto foi iniciado, e os dados começam a ser armazenados. O gesto é encerrado quando não há variação de magnitude grande o suficiente no vetor de aceleração após alguns instantes.

Este método não exige que o usuário pressione botões para indicar início e fim de gesto, mas pode gerar inúmeros problemas de interpretação. Um movimento de simples posicionamento do usuário pode ser interpretado erroneamente como parte de um gesto. Também é necessário aumentar os intervalos entre as chamadas de getGesture, para permitir que o usuário volte à sua posição inicial para poder reconhecer um novo gesto. Foi considerada uma abordagem similar à usada no módulo do Wiimote, mas os botões do SPOT são muito pequenos e difíceis de pressionar. Existem métodos mais sofisticados de determinar início e fim de gesto, mas este trabalho não tem como objetivo compará-los nem criar melhores. Por isso foi adotado o método exposto, considerado mais simples.

Tanto os SPOTs quanto o Wiimote possuem o mesmo tipo de acelerômetro. Por isso, foram aplicadas as mesmas técnicas de filtragem e quantização nos dois módulos. As únicas diferenças são o processo de calibração (no Wiimote ele não é necessário, pois os dados de calibração vêm armazenados no próprio controle) e a taxa de amostragem, que no SPOT é um pouco maior.

3.3.8 NeuralNetRecognitionModule: um módulo de reconhecimento que usa Redes Neurais

O NeuralNetRecognitionModule (ou NNetRM) é o segundo módulo de dispositivo criado neste trabalho. O módulo usa redes neurais previamente treinadas para classificar gestos.

Foi criada uma FFNN (ou percéptron de múltiplas camadas) com uma camada de entrada, uma camada intermediária ou oculta com 15 neurônios (quantidade determinada empiricamente) e uma camada de saída com o mesmo número de neurônios que o número de gestos que a rede é capaz de reconhecer. Para determinar o número de neurônios da camada de entrada é necessário criar uma quantidade de neurônios para processar as informações vindas de cada instante do movimento, ou seja, teríamos $Q \times T$ neurônios de entrada. Nosso objetivo então é estipular valores para as duas variáveis.

A variável T representa o comprimento máximo do gesto em instantes. Teoricamente, um gesto pode ser tão longo quanto se queira, portanto não há como determinar um valor fixo que sirva para qualquer gesto. Na etapa de treinamento, pode-se buscar pelo maior valor de T dentre todas as execuções do gesto:

$$T = max_{1 \leq i \leq w}\{|\epsilon_i|\}, \text{ para todo } \sigma_x. \tag{3.2}$$

Esta estratégia foi adotada; entretanto, isso ainda não garante que não ocorra um valor maior que T no momento do reconhecimento. Por isso as informações de instantes posteriores a T são descartadas no momento do reconhecimento.

A variável Q representa o número de entradas que a rede recebe em um instante t_i. O ideal seria que Q fosse igual ao número de observações distintas provenientes do dispositivo, isto é, $Q = |\Psi|$, e o conjunto I de entradas da rede seria construído de acordo com a seguinte fórmula:

$$I_{Qt+q} = \begin{cases} 1 & , \text{ se } \psi_q \in \iota_t \\ 0 & , \text{ caso contrário} \end{cases} \tag{3.3}$$

Essa abordagem permitiria comunicar todas as observações possíveis em cada instante usando variáveis booleanas, mas para valores de T altos criaria redes com um número excessivo de entradas. A alternativa que usaria o menor número de entradas definiria o conjunto I de acordo com a seguinte fórmula:

$$I_t = \psi_q, \text{ se } \psi_q \in \iota_t \tag{3.4}$$

Com esta abordagem a rede teria apenas T entradas, mas dois problemas seriam criados. Como desta forma só há uma observação por instante, é perdido o suporte a múltiplas observações por instante ($|\iota_x| = 1$). Além disso, valores de ψ_q maiores gerariam estímulos maiores na entrada da rede; isto viola a independência entre rótulos, porque faz os rótulos de maior índice terem maior influência sobre a rede.

Foi escolhida uma abordagem intermediária, que não permite múltiplas observações por instante, mas usa variáveis booleanas como entrada e não gera um número muito grande de neurônios. A ideia é, ao invés de termos uma entrada (ψ_q) no instante t como no caso anterior, termos a representação de ψ_q em binário:

$$\begin{aligned} Q &= \lfloor (log_2 |\Psi| + 1) \rfloor \\ D_{\psi_t} &= \{b_1, b_2, ..., b_Q\} \\ I_{Qt+q} &= D_{\psi_t}\{b_q\} \end{aligned} \tag{3.5}$$

Aqui o conjunto D representa a decomposição de ψ_t em binário, e os b_i são os bits da representação. Então, para determinar o valor da entrada de índice $Qt + q$, é feita a decomposição de ψ_t e usado o bit b_q. Como Q é consideravelmente menor, a rede possui bem menos entradas.

Para a criação das redes neurais do módulo foi testado o Joone ([Joo]), um arcabouço modular que permite a escolha livre dos componentes que integram a rede. Os tipos das camadas de neurônios e das sinapses podem ser facilmente trocados, originando redes completamente diferentes com poucas mudanças em código.

A classe GestureManager gerencia a rede neural, os dados de treinamento e as associações entre nomes de gestos e as saídas da rede. Ao contrário dos HMMs, uma única rede contém todas as informações para reconhecimento de todos os gestos. Quando um novo gesto é treinado, a rede anterior é destruída e é criada uma nova, que é treinada com os dados de TODOS os gestos já registrados acrescidos dos dados do novo gesto. A rede nova possuirá o número de saídas da rede anterior mais um, uma saída para cada gesto. O treinamento é feito com o algoritmo de retropropagação de erro (*error backpropagation algorithm*), com número de ciclos, taxa de aprendizado e momento fixos.

Para reconhecer um gesto, as suas observações são decompostas na forma binária e usadas como entrada da rede. O sinal é propagado na rede através de *camadas sigmoidais* até chegar às saídas; isso garante que o valor de cada saída esteja no intervalo [0,1], representando a probabilidade dos

dados de entrada ser uma instância do gesto representado pela saída. A saída com valor máximo representa o gesto mais provável; se esse valor estiver abaixo de um limite de tolerância de erro, o método recognize devolve *null*.

Capítulo 4

Testes e Experimentos

Conceitualmente, para validar a arquitetura proposta, bastaria iniciar a execução do GRMediator com um módulo de dispositivo e um de reconhecimento, e efetuar a troca de algum deles durante a execução. Se o programa for capaz de continuar a execução normalmente após a troca, isto é, ainda ser capaz de efetuar treinamento e reconhecimento com os novos módulos, então o objetivo terá sido validado. Porém, é também necessário saber o quão preciso é o reconhecimento nesta arquitetura, pois se não conseguirmos reconhecer nenhum gesto ela não será aplicável na prática. Esta seção será portanto dividida em duas partes.

4.1 Testes de troca de módulos

Existem algumas regras para a troca de módulos, que são reforçadas pelo mediador.

- Se estivermos tentando trocar um módulo de dispositivo no qual já invocamos startDevice(), o mediador invoca o método stopDevice() para garantir que a comunicação com o dispositivo atual seja encerrada antes de usarmos outro.
- Ao efetuarmos treinamento com um par de módulos, os dados armazenados pelo módulo de reconhecimento só farão sentido com o módulo de dispositivo que forneceu aqueles dados. Portanto, com a mudança de algum dos módulos, é importante invocar o método clear() para garantir que o novo módulo não use informações do módulo antigo. Isto implica que, ao trocarmos um módulo, temos que treinar o sistema novamente com o novo módulo se quisermos reconhecer gestos. Por isso é recomendado salvar arquivos com dados de treinamento (através do método save()) e somente carregá-los quando necessário.

4.1.1 Troca de dispositivo

```
1
2 Mediator m = new Mediator();
3 m.setDeviceModule("WDM.jar");
4 m.setReconModule("HMMRM.jar");
5 m.startDevice();
6 System.out.println("Treine o primeiro gesto.");
7 m.train("g1", 5);
8 System.out.println("Treine o segundo gesto.");
9 m.train("g2", 5);
10 for (int i = 0; i < 10; i++) {
11   System.out.println(m.recognize());
12 }
13 m.stopDevice();
14
15 m.setDeviceModule("SPOTDM.jar");
16 m.startDevice();
17 System.out.println("Treine o primeiro gesto.");
18 m.train("g1", 5);
19 System.out.println("Treine o segundo gesto.");
20 m.train("g2", 5);
21 for (int i = 0; i < 10; i++) {
22   System.out.println(m.recognize());
23 }
24 m.stopDevice();
```

O código exibido instancia um mediador, escolhendo como dispositivo o Nintendo Wiimote e como modelo matemático os HMMs. Ele então inicializa o dispositivo e pede que sejam treinados dois gestos, “g1” e “g2”. Cada um deles é feito 5 vezes, e o conjunto de dados resultante é usado para treinar o modelo matemático. Em seguida o sistema reconhece 10 gestos feitos pelo usuário, imprimindo na tela o nome do gesto efetuado ou *null* se ele não foi reconhecido. Após o último gesto o dispositivo é desligado.

No início do segundo bloco de código é simulada a troca de dispositivo para o Sun SPOT. É importante notar que o próprio mediador se encarrega neste momento de excluir os dados de reconhecimento armazenados para evitar inconsistências; portanto é necessário treinar o sistema novamente com o novo dispositivo. O procedimento é análogo ao do primeiro dispositivo.

A execução do código gerou o resultado esperado, e o sistema foi capaz de trabalhar com o segundo dispositivo como se o primeiro nunca tivesse sido usado antes dele.

4.1.2 Troca de modelo matemático

```
1
2 Mediator m = new Mediator();
3 m.setDeviceModule("WDM.jar");
4 m.setReconModule("HMMRM.jar");
5 m.startDevice();
6 System.out.println("Treine o primeiro gesto.");
7 m.train("g1", 5);
8 System.out.println("Treine o segundo gesto.");
9 m.train("g2", 5);
10 for (int i = 0; i < 10; i++) {
11   System.out.println(m.recognize());
12 }
13
14 m.setReconModule("NNetRM.jar");
15 System.out.println("Treine o primeiro gesto.");
16 m.train("g1", 5);
17 System.out.println("Treine o segundo gesto.");
18 m.train("g2", 5);
19 for (int i = 0; i < 10; i++) {
20   System.out.println(m.recognize());
21 }
22 m.stopDevice();
```

O código exibido faz exatamente o mesmo do exemplo anterior até o fim do primeiro bloco, com a exceção de que não interrompe o dispositivo para efetuar a troca de modelo matemático; afinal, isso não é necessário graças à independência entre os módulos. No segundo bloco o sistema é configurado para usar Redes Neurais como modelo matemático; nesse ponto o sistema também exclui as informações prévias de gestos por segurança. Em seguida o sistema é re-treinado e são reconhecidos 10 gestos da mesma forma.

A execução do código gerou o resultado esperado, e o sistema foi capaz de trabalhar com o segundo modelo matemático como se o primeiro nunca tivesse sido usado antes dele.

4.1.3 Troca de ambos

```
1
2 Mediator m = new Mediator();
3 m.setDeviceModule("WDM.jar");
4 m.setReconModule("HMMRM.jar");
5 m.startDevice();
6 System.out.println("Treine o primeiro gesto.");
7 m.train("g1", 5);
8 System.out.println("Treine o segundo gesto.");
9 m.train("g2", 5);
10 for (int i = 0; i < 10; i++) {
11   System.out.println(m.recognize());
12 }
13 m.stopDevice();
14
15 m.setDeviceModule("SPOTDM.jar");
16 m.startDevice();
17 System.out.println("Treine o primeiro gesto.");
18 m.train("g1", 5);
19 System.out.println("Treine o segundo gesto.");
20 m.train("g2", 5);
21 for (int i = 0; i < 10; i++) {
22   System.out.println(m.recognize());
23 }
24
25 m.setReconModule("NNetRM.jar");
26 System.out.println("Treine o primeiro gesto.");
27 m.train("g1", 5);
28 System.out.println("Treine o segundo gesto.");
29 m.train("g2", 5);
30 for (int i = 0; i < 10; i++) {
31   System.out.println(m.recognize());
32 }
33 m.stopDevice();
```

O código exibido conjuga os dois exemplos anteriores, primeiro trocando o dispositivo para o Sun SPOT e depois trocando o modelo matemático para Redes Neurais. A sua execução gerou o resultado esperado, mais uma vez demonstrando a independência entre os módulos.

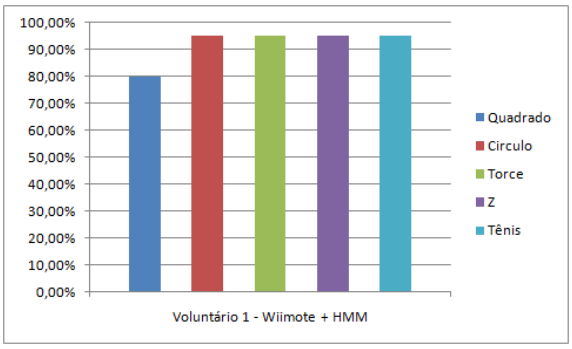
4.2 Comparação de acuidade de reconhecimento

Foram escolhidos testes similares aos feitos em [SPHB08], especialmente porque os módulos deste trabalho são adaptações de seu código-fonte. Desta forma pode-se medir o impacto da aplicação da interface sobre a sua implementação. O conjunto de gestos escolhido é o mesmo usado como referência no trabalho deles; assim pode-se aproveitar a taxa média de reconhecimento que eles obtiveram como resultado.

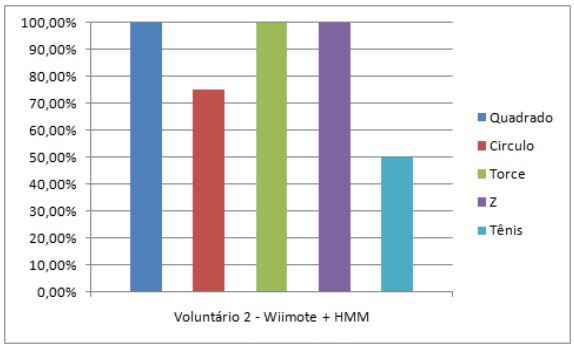
Schlömer *et.al.* efetuaram testes com 6 participantes. Cada participante foi apresentado a 5 gestos através de desenhos, e informados que teriam que executar cada um dos gestos 15 vezes, totalizando 75 execuções. Os gestos são: "quadrado" (um quadrado desenhado no ar), com reconhecimento médio de 88,8%; "círculo" (um círculo desenhado no ar), com taxa de 86,6%; "torce" (consiste em torcer o pulso 90 graus para a direita e voltar à posição original), com taxa de 84,3%; "Z" (uma letra Z desenhada no ar, de cima para baixo), com taxa de 94,3%; e "tênis" (um movimento de saque de partidas de tênis, levantando o controle acima da cabeça e abaixando-o rapidamente), com taxa de 94,5%. Os testes utilizam o Nintendo Wiimote como dispositivo e HMMs como modelo matemático. Estes dados estão disponíveis no artigo original [SPHB08].

Neste trabalho foi adotada uma abordagem similar, sendo efetuados testes com 5 voluntários. Cada um dos voluntários efetuou cada um dos gestos supracitados 20 vezes com o intuito de treinar o sistema, totalizando uma carga de treinamento de 100 gestos. Em seguida, cada um dos gestos foi efetuado novamente por 20 vezes, desta vez com o intuito de classificação. Nestes testes também foi utilizado o Nintendo Wiimote como dispositivo e HMMs como modelo matemático. Foram registradas as classificações que o sistema atribuiu a cada gesto, e gerados gráficos exibindo a acuidade de classificação da biblioteca para aquele participante. (Veja as figuras 4.1(a), 4.1(b), 4.1(c), 4.1(d) e 4.1(e)) Foi também criado um gráfico com a média para os 5 participantes (Figura 4.2).

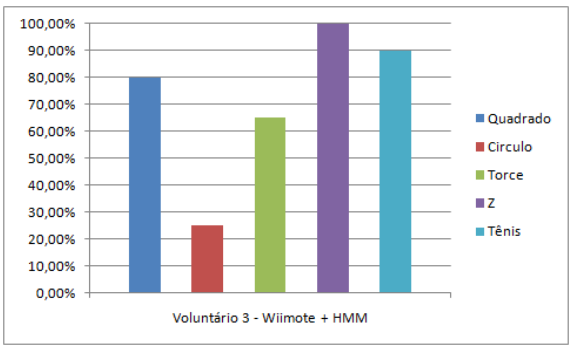
A média de reconhecimento obtida indica que, apesar de oferecer uma performance aceitável, houve alguma perda de precisão de reconhecimento em comparação com a média obtida no trabalho de Schlömer *et.al.* Os dois trabalhos usam o mesmo dispositivo e modelo matemático, e efetuam as mesmas etapas de pré-processamento dos dados, exceto por um fator extra encontrado no trabalho original: a quantização customizada. Ao treinar um gesto, é criado um *codebook* que minimiza as distâncias dos vetores de aceleração obtidos até os centróides, e depois esse *codebook* é usado no momento de classificação para gerar uma quantização mais adequada àquele gesto. A quantização customizada não foi implementada neste trabalho porque o quantizador faz parte do módulo de dis-



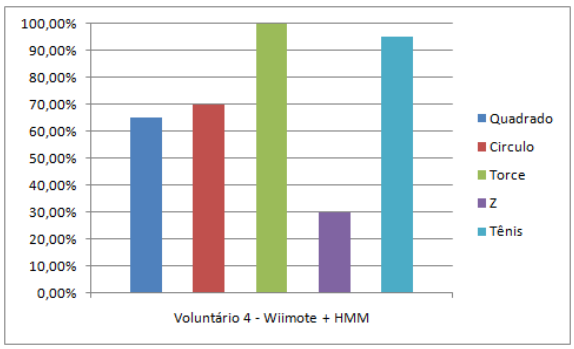
(a) Porcentagens de acerto para o voluntário 1.



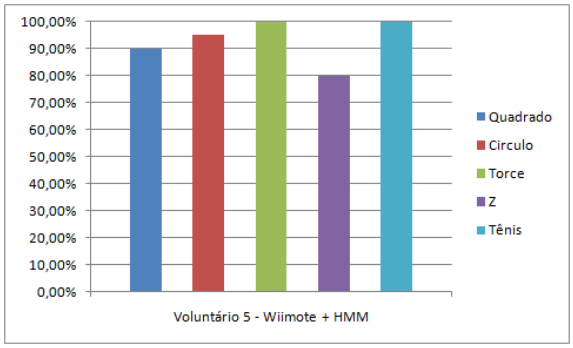
(b) Porcentagens de acerto para o voluntário 2.



(c) Porcentagens de acerto para o voluntário 3.



(d) Porcentagens de acerto para o voluntário 4.



(e) Porcentagens de acerto para o voluntário 5.

Figura 4.1: *Comparação de acuidade de reconhecimento com 5 voluntários (Wiimote + HMM)*

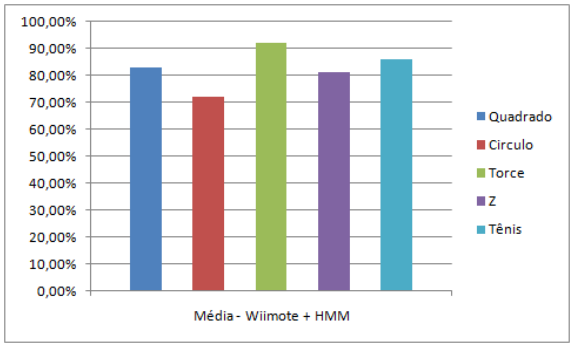
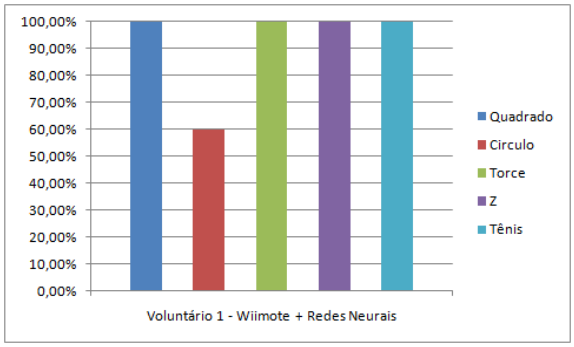


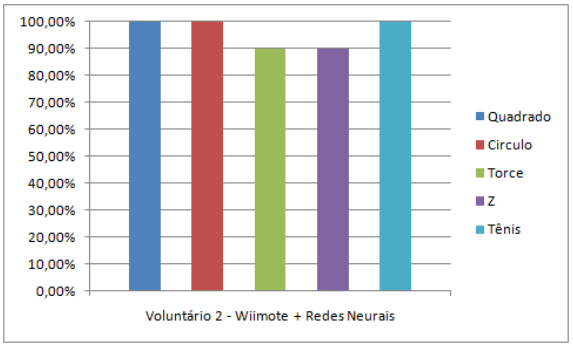
Figura 4.2: *Média do reconhecimento com 5 voluntários (Wiimote + HMM)*

positivo e os dados de treinamento são armazenados no módulo de reconhecimento; esse mecanismo exigiria que o módulo de dispositivo também armazenasse os dados de treinamento para poder criar o *codebook* customizado, o que adicionaria complexidade desnecessária à arquitetura.

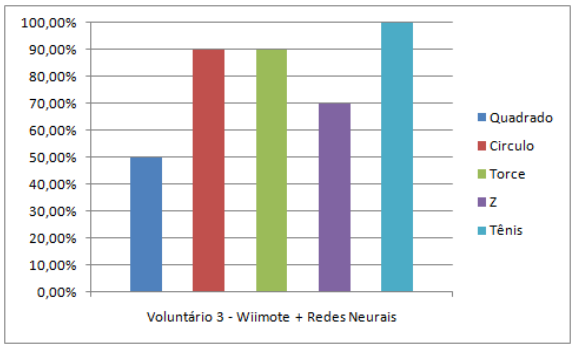
Os testes seguintes usam Redes Neurais como modelo matemático. Para cada um dos 5 participantes do experimento anterior, foi carregado o conjunto de 100 gestos efetuados no treinamento dos HMMs. Das 20 execuções de cada gesto, 10 foram usadas para treinar as redes, e as outras 10 foram usadas para efetuar classificação automática, gerando um total de 50 classificações por voluntário. Foram então gerados gráficos exibindo a acuidade de classificação da biblioteca para cada participante. (Veja as figuras 4.3(a), 4.3(b), 4.3(c), 4.3(d) e 4.3(e)) Foi também criado um gráfico com a média para os 5 participantes (Figura 4.4).



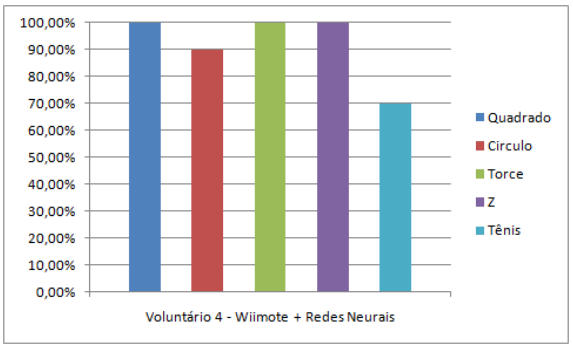
(a) Porcentagens de acerto para o voluntário 1.



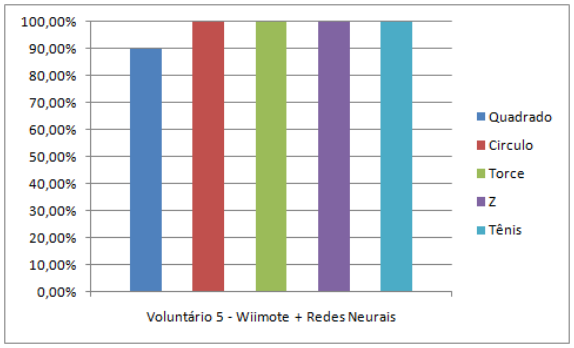
(b) Porcentagens de acerto para o voluntário 2.



(c) Porcentagens de acerto para o voluntário 3.



(d) Porcentagens de acerto para o voluntário 4.



(e) Porcentagens de acerto para o voluntário 5.

Figura 4.3: *Comparação de acuidade de reconhecimento com 5 voluntários (Wiimote + Redes Neurais)*

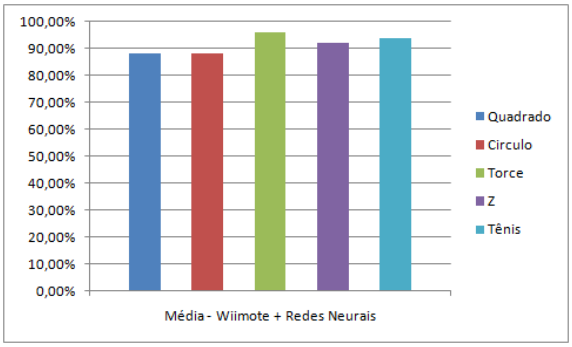
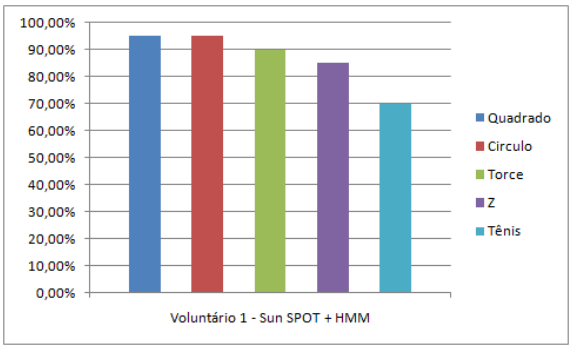


Figura 4.4: *Média do reconhecimento com 5 voluntários (Wiimote + Redes Neurais)*

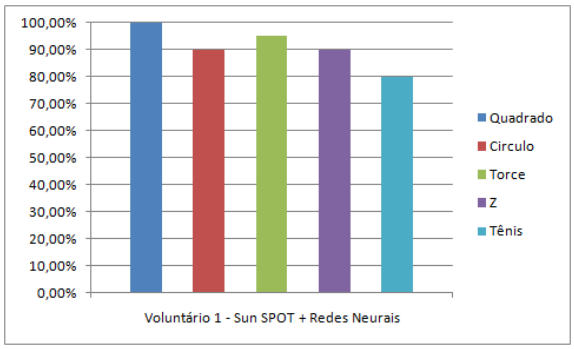
Houve uma melhora perceptível no reconhecimento em relação aos HMMs implementados. As redes neurais apresentaram um tempo de treinamento muito superior, que aumenta consideravelmente em proporção ao tamanho do conjunto de treinamento, mas mostraram uma capacidade de generalização maior em relação aos HMMs, mesmo com um número menor de amostras.

Um ponto em comum dos dois modelos matemáticos foi o fato de se confundirem com maior frequência entre os gestos "quadrado" e "círculo". Além de serem parecidos, a quantização dos dados do Wiimote não leva em consideração as paradas do controle (momentos de repouso não são quantizados), um aspecto que ajuda a diferenciar os dois gestos.

Os últimos testes, realizados com os Sun SPOTs, são mostrados nas figuras 4.5(a) e 4.5(b). Como o mecanismo de operação dos SPOTs é mais delicado e como seu módulo ainda apresentava alguns problemas na comunicação por rádio, os testes foram realizados com somente um voluntário.



(a) Sun SPOTs + HMM



(b) Sun SPOTs + Redes Neurais

Figura 4.5: *Acuidade de reconhecimento de 1 voluntário com o Sun SPOT.*

Há pouca informação para se tirar conclusões a respeito dos Sun SPOTs. A tendência é que, com o aprimoramento do módulo, os SPOTs fiquem muito similares ao Wiimote, visto que ambos possuem o mesmo acelerômetro.

Capítulo 5

Conclusões

A arquitetura apresentada conseguiu atingir o nível de independência entre dispositivo e modelo matemático desejado. O mediador, camada que faz a comunicação entre os módulos, garante que não hajam referências diretas entre eles, possibilitando o fácil intercâmbio desses componentes. Para começar a usar um outro dispositivo ou modelo matemático, basta excluir todas as informações de treinamento armazenadas e efetuar um novo treinamento, ou carregar dados de um treinamento anterior. Essa exclusão garante que o sistema não confunda os rótulos recebidos de um dispositivo com os de outro, cujos significados, embora desconhecidos para o sistema, podem ser completamente diferentes. Essa independência entre rótulos exige que o modelo matemático assuma que todo rótulo obtido na comunicação entre os módulos seja independente dos outros; essa hipótese é suficiente para trabalhar com dispositivos como acelerômetros e luvas, desde que escolhida uma quantização adequada. Como discutido, não há uma forma de determinar um algoritmo de quantização que sirva para todo dispositivo, pois cada um deles define quais de suas medidas são relevantes ao reconhecimento de gestos, e como são transmitidas. Por isso, determinar uma boa quantização é uma tarefa experimental e depende de interação com módulos de reconhecimento já criados.

Quanto à acuidade de reconhecimento, a arquitetura proposta, por necessariamente efetuar quantização para determinar os rótulos, abre mão de alguma precisão quando comparada com soluções que fazem processamento direto dos dados do dispositivo no modelo matemático. Esse era um efeito esperado, mas os experimentos mostraram que a perda não foi muito significativa, e com algumas melhoras nos módulos e na arquitetura é possível que essa diferença diminua ainda mais.

É importante notar que, independente de qual solução se adote para o problema de reconhecimento de gestos, o sistema sempre terá maior dificuldade de classificar gestos muito parecidos. Quanto mais característico for cada gesto do conjunto a reconhecer, melhor tende a ser a precisão

do sistema, ou seja, o conjunto também influencia nos resultados. Por isso, não se pode esperar uma taxa de reconhecimento elevada para qualquer conjunto de gestos.

5.1 Trabalhos Futuros

Existem vários pontos onde este trabalho pode ser melhorado ou estendido. Aqui são discutidos alguns destes pontos.

5.1.1 Melhora de módulos existentes

Os módulos criados neste trabalho tiveram como principal objetivo servir como prova de conceito para a interface proposta. Entretanto, há algumas melhoras que podem ser feitas em cada um deles para que, usados em conjunto, seja obtido um reconhecimento mais preciso.

Uma possível melhora é a redução da avidez dos módulos de reconhecimento, isto é, garantir que o módulo dê uma resposta *null* para gestos que não correspondam a nenhum dos treinados. Por exemplo, sem nenhuma verificação sobre o tamanho do conjunto de dados a reconhecer, o módulo de HMMs pode facilmente classificar gestos de poucas observações como sendo instâncias de gestos muito mais compridos. Isto acontece porque o modelo utilizado (de Bakis) faz com que as probabilidades rapidamente concentrem-se no estado final do HMM, já devolvendo probabilidades altas para gestos cujo número de observações alcança o número de estados do HMM, e dando menos relevância às observações posteriores.

Este problema pode ser mitigado com a mudança para um modelo alternativo de HMM, que considere mais possibilidades de transições entre os estados, ou com um mecanismo de inferência automática de estados, que atribua o número de estados do HMM proporcionalmente ao tamanho do gesto, ao invés de um número fixo. Assim, gestos mais curtos serão naturalmente associados com maior probabilidade a modelos com menor número de estados.

5.1.2 Criação de novos módulos

Somente há evidência teórica quanto à adequação da interface proposta a outros dispositivos e modelos matemáticos. Novos módulos podem fornecer evidência empírica, ou revelar a necessidade de modificação da interface para funcionar adequadamente. Em particular, o suporte a módulos que trabalham com múltiplas observações por instante deve ser testado e ajustado se necessário.

Um dispositivo interessante para testar a arquitetura é a luva com acelerômetro. Como já mencionado, as leituras do acelerômetro e as flexões dos dedos são independentes, o que geraria

mais de uma observação por instante. Entretanto, para poder testar o módulo desse dispositivo seria necessário criar um módulo de reconhecimento que também ofereça suporte a múltiplas observações; HMMs adaptados e TDNNs seriam bons candidatos a modelos matemáticos para tal módulo. Outra possibilidade é a criação de um módulo que use câmeras como dispositivo; em teoria, com uma boa quantização, ele também poderia ser usado. A implementação forneceria evidência sobre a viabilidade da interface para uso com dispositivos extrusivos.

5.1.3 Uso de arquivos de configuração para customizar módulos

É comum que componentes complexos forneçam alguma forma de modificar seus parâmetros de funcionamento sem obrigar a sua reprogramação; tais componentes tornam-se mais fáceis de utilizar. Uma forma bastante encontrada envolve o uso de arquivos de configuração. Em tais arquivos são especificados os parâmetros que podem mudar o comportamento do componente durante sua execução, e ao ser iniciado o componente acessa esses arquivos buscando por esses parâmetros para configurar-se corretamente.

Os componentes mostrados neste trabalho (aqui chamados de módulos) não são muito complexos, mas precisam ser reprogramados caso deseje-se mudar algum aspecto particular de seu comportamento (por exemplo, mudar o número de estados dos HMMs ou o número de ciclos de treinamento das redes neurais), informações pequenas que consistem em um valor de uma variável ou na seleção de um determinado algoritmo. Tais informações poderiam ser passadas como parâmetro para esses componentes através de arquivos de configuração, assim eliminando a necessidade de reprogramação para essas mudanças pequenas. Os módulos seriam então alterados para buscar por tais informações nesses arquivos.

5.1.4 Transmissão de relacionamentos entre rótulos

Na arquitetura apresentada neste trabalho, foi mostrado (na seção 1.2) que é importante que o módulo de reconhecimento assuma que todos os possíveis rótulos emitidos pelo módulo de dispositivo sejam independentes, porque as relações entre os rótulos podem variar de um dispositivo para outro. Entretanto, se o módulo de reconhecimento tivesse tais informações, ele poderia usá-las para efetuar um reconhecimento mais preciso (de acordo com o exemplo exibido na mesma seção).

Como essas informações mudam de acordo com o dispositivo, uma possibilidade de integrá-las à arquitetura seria adicionar uma operação ao módulo de dispositivo que devolve as relações entre rótulos, e uma operação ao módulo de reconhecimento que recebe e armazena essas relações. Ao

carregar um módulo, essas operações seriam invocadas automaticamente, assim garantindo que as relações sejam levadas em conta no treinamento e no reconhecimento.

Consideremos novamente o dispositivo D1 da seção 1.2. Se tivéssemos o mecanismo em questão, o módulo de dispositivo poderia informar ao módulo de reconhecimento que o rótulo “r1” é mais próximo dos rótulos “r2”, “r4” e “r5” do que os outros. Assim, ao classificar um gesto, o módulo de reconhecimento pode determinar se um conjunto de observações é mais ou menos distante do modelo treinado sem precisar das informações sobre variações comuns, presentes no conjunto de treino. Usar as relações entre rótulos e variações comuns simultaneamente pode oferecer um reconhecimento bem mais robusto.

Referências Bibliográficas

[Bor08] Paul-Valentin Borza. Motion-based gesture recognition with an accelerometer. Dissertação de Mestrado, Babe-Bolyai University of Cluj-Napoca, Romania - Faculty of Mathematics and Computer Science, 2008. 1, 2, 12

[CMZ05] M. C. Cabral, C. H. Morimoto, e M. K. Zuffo. On the usability of gesture interfaces in virtual reality environments. *Proceedings of the 2005 Latin American conference on Human-computer interaction*, páginas 100–108, 2005. 2

[E+03] J. Eisestein et al. Device independence and extensibility in gesture recognition. *Proceedings of the IEEE Virtual Reality Conference*, página 207, 2003. 21, 26

[EKR98] S. Eickeler, A. Kosmala, e G. Rigoll. Hidden markov model based continuous online gesture recognition. *14th International Conference on Pattern Recognition (ICPR'98)*, 1998. 2, 12, 34

[FR94] W. T. Freeman e M. Roth. Orientation histograms for hand gesture recognition. Em *In International Workshop on Automatic Face and Gesture Recognition*, páginas 296–301, 1994. 2

[FSS98] C. Faisstnauer, D. Schmalstieg, e Z. Szalavri. Device-independent navigation and interaction in virtual environments. *VRAIS'98*, 1998. 21, 26

[GHJV95] E. Gamma, R. Helm, R. Johnson, e J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addinson-Wesley, 1995. 42

[Har93] P. A. Harling. Gesture input using neural networks. Dissertação de Mestrado, University of York, 1993. 1, 14

[Hay05] S. Haykin. *Neural Networks - A Comprehensive Foundation*. Pearson Education, 2º edição, 2005. 15

[HK93] T. He e A. E. Kaufman. Virtual input devices for 3d systems. *Proceedings of the IEEE Conference on Visualization*, páginas 142–148, 1993. 21, 26

[JMM96] A. K. Jain, Jianchang Mao, e K. M. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer Society Magazine, Volume 29, Issue 3*, páginas 31–44, 1996. 13, 14, 17

[Joo] Joone: Java object-oriented neural engine. http://sourceforge.net/projects/joone/. Último acesso em 07/10/10. 54

[KSL07] L. Kratz, M. Smith, e F. J. Lee. Wiizards: 3d gesture recognition for game play input. *Proceedings of the 2007 conference on Future Play*, páginas 209–212, 2007. 1, 2, 12

[Män01] Vesa-Matti Mäntylä. Discrete hidden markov models with application to isolated user-dependent hand gesture recognition. Dissertação de Mestrado, Technical Research Centre of Finland, 2001. 12, 16, 23, 51

[MT91] K. Murakami e H. Taguchi. Gesture recognition using recurrent neural networks. *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, páginas 237–242, 1991. 1, 14

[Rab89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, páginas 257–286, 1989. 11, 12, 16, 17, 50

[San97] A. Sandberg. Gesture recognition using neural networks. Dissertação de Mestrado, 1997. 1, 14

[SP03] C. Szyperski e C. Pfister. Workshop on component-oriented programming, summary. Em *Special Issues in Object-Oriented Programming - ECOOP96 Workshop Reader*. Springer-Verlag, 2003. 19

[SPHB08] T. Schlömer, B. Poppinga, N. Henze, e S. Boll. Gesture recognition with a wii controller. *Proceedings of the 2nd international conference on Tangible and embedded interaction*, páginas 11–14, 2008. 1, 2, 8, 9, 10, 12, 38, 48, 49, 50, 61

[Sun] Sun spot world. http://www.sunspotworld.com/. Último acesso em 07/10/10. 51

[SZP07] S. Sreedharan, E. S. Zurita, e B. Plimmer. 3d input for 3d worlds. *Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces*, páginas 227–230, 2007. 1

[Szy02] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 2º edição, 2002. 19

[WC01] K. Daniel Wong e Donald C. Cox. Two-state pattern-recognition handoffs for corner-turning situations. *IEEE Transactions on Vehicular Technology*, páginas 354–363, 2001. 34

[wii] Wiibrew. http://wiibrew.org/wiki/Wiimote. Último acesso em 07/10/10. 1, 48

[YA99] Ming-Hsuan Yang e Narendra Ahuja. Recognizing hand gesture using motion trajectories. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, páginas 466–472, 1999. 2, 14