Universidade Atlântica

Licenciatura em Gestão de Sistemas e Computação

Visão Computacional:

Reconhecimento da Mão Humana e Seus Movimentos

Trabalho final para Laboratório de GSC Aplicado

Elaborado por Tiago Manuel Saraiva Marques

Aluno nº 20111501

Orientador: Professor Doutor Sérgio Nunes

Coorientador: Professor Doutor Mário Macedo

Barcarena, Novembro de 2014

Visão Computacional:

Reconhecimento da Mão Humana e Seus Movimentos

Trabalho final para Laboratório de GSC Aplicado

Aluno: Tiago Manuel Saraiva Marques

Aluno nº 20111501

Orientador: Professor Doutor Sérgio Nunes

Coorientador: Professor Doutor Mário Macedo

Barcarena, Novembro de 2014

O autor é o único responsável pelas ideias expressas neste relatório.

Agradecimentos

Este percurso académico que se encerra agora com a apresentação deste trabalho não teria sido possível sem a ajuda, compreensão e apoio de algumas pessoas, que me apoiaram direta ou indiretamente nesta etapa e para as quais eu gostaria de deixar um agradecimento especial.

À minha família, em especial à minha esposa e filha, pelo seu apoio emocional e compreensão, sem os quais não seria de todo possível concluir este projeto;

Ao meu amigo, colega e parceiro nesta etapa, Hugo Ferreira, pelo seu apoio e companheirismo que estiveram presentes desde o primeiro dia;

À turma de Gestão de Sistemas e Computação, pelo companheirismo e amizade;

À Universidade Atlântica e ao seu corpo docente por me possibilitarem evoluir como ser humano e me transmitirem conhecimentos que me acompanharão ao longo da minha vida;

Ao Professor Mário Macedo, coorientador deste trabalho, tenho a agradecer, pela sua orientação, disponibilidade e ajuda;

Ao Professor Doutor Sérgio Nunes, meu orientador, por todo o apoio dado neste projeto, pela sua orientação e disponibilidade e por ter sido um excelente professor ao longo de toda a licenciatura. Sem a sua ajuda e orientação, este trabalho não teria sido possível.

Resumo

O presente trabalho tem como principal objetivo estudar a visão computacional, nomeadamente no que diz respeito ao reconhecimento da mão humana e seus movimentos.

Assim, propõe-se tentar perceber como interagir com a tecnologia do dia-a-dia através de um interface natural, o movimento da mão.

Além da revisão bibliográfica acerca do tema e da metodologia a aplicar, irá ser apresentado um protótipo, com recurso ao sensor Kinect da Microsoft versão 1, que será capaz de reconhecer a mão humana, de a seguir e reconhecer o movimento de abrir e fechar a mão.

Palavras-chave: Visão Computacional, Sensor Kinect, Interface Natural.

Abstract

This work main objective is to study the computer vision field, especially the work related to human hand recognition and its movements.

It proposes to find out how to interact with the everyday technology through a natural interface like the hand movement.

Besides the literature review and the methodology on the subject, this work includes the development of a prototype, using Microsoft Kinect version 1, to recognize and track the hand movement and also be able to identify if it is opened or closed.

Keywords: Computer Vision, Kinect Sensor, Natural User Interface.

Índice

VOLUME I

Índice de figuras

Introdução

O presente trabalho reflete o culminar do meu percurso enquanto estudante de licenciatura em Gestão de Sistemas e Computação.

Após ponderar alguns temas do meu interesse pessoal, optei por abordar a visão computacional, nomeadamente no que diz respeito ao reconhecimento da mão humana e seus movimentos.

A área da visão computacional sempre foi um vasto e complexo ramo da computação. Desde as ferramentas de edição de imagem, ao reconhecimento de texto, à deteção de movimento, ao reconhecimento facial, entre outros, esta é talvez uma das mais vastas e complexas temáticas da computação atual e futura.

Os investigadores que trabalham esta temática, têm encontrado inúmeros desafios e limitações, mas também soluções e/ou alternativas para as mesmas.

De facto, existem ainda limitações para as quais não existe solução à vista. Mas é também verdade que, à medida que a tecnologia progride, várias são ultrapassadas.

Dos vários problemas que a visão computacional apresenta, este trabalho pretende focar-se na área do reconhecimento de imagens, nomeadamente da mão humana. Para tal procurarei apresentar as atuais abordagens a esta temática, bem como as possíveis soluções.

Em termos de organização estrutural do trabalho, o mesmo terá três capítulos.

O primeiro capítulo aborda a revisão bibliográfica da área da Visão Computacional, as suas problemáticas, bem como as metodologias inerentes à área, nomeadamente na identificação, segmentação e seguimento de corpos ou objetos num cenário captado por uma câmara digital.

Tiago Marques
Novembro de 2014
Universidade Atlântica

O segundo capítulo explica a abordagem do professor Davison ao problema de reconhecimento da mão através de Visão Computacional, neste caso sem recurso a sensores de profundidade e conseguindo a segmentação da mão através do contraste, conseguindo inclusive identificar os dedos da mão. Este exemplo é pertinente pois demonstra os princípios referidos na revisão bibliográfica, nomeadamente os de segmentação e seguimento (*tracking*) da mão sem recurso a outros meios que não a imagem captada por uma simples câmara.

O terceiro capítulo trata a abordagem prática do desenvolvimento de um protótipo com recurso ao sensor Kinect. Este protótipo será capaz de reconhecer a mão e os seus movimentos e proporcionar assim um interface natural com a máquina.

Ainda neste último capítulo são demonstrados uma série de testes ao protótipo e com base nesses testes são apresentadas as limitações do mesmo.

Problema e Motivação

Desde a “explosão” das novas tecnologias, que se iniciou na segunda metade do século XX, que se pode verificar que o ser humano procura ter, de forma cómoda, o máximo de controlo sobre os dispositivos que o rodeiam.

Podemos verificar isto em várias situações do quotidiano como por exemplo o acender a televisão com o comando de infravermelhos, ou até mais recentemente com o telemóvel; o abrir a porta da garagem com um comando, entre muitos outros exemplos.

Para atingir esta finalidade, ao longo da História, vários inventores têm vindo a desenvolver interfaces Homem-Máquina, que estabelecem a ponte entre o utilizador e a tecnologia. A maneira como o comando da televisão funciona, como os seus botões estão organizados, como estão legendados e como os menus da televisão são estruturados é um exemplo de um interface estabelecido para que possa ser possível ao humano controlar facilmente a máquina.

É interessante a ideia de que se podem desenvolver vários tipos de interfaces e que não é necessário qualquer tipo de contacto com equipamentos físicos para estabelecer este interface e controlar a máquina.

Já há algum tempo que os avanços na visão computacional nos levam no caminho de possibilitar ao ser Humano controlar a tecnologia com apenas os seus movimentos, embora o processo computacional não seja simples, mas sim extremamente complexo. Tal facto leva a que seja necessário “pesar” a complexidade dos algoritmos de deteção e seguimento de movimentos, uma vez que, para aumentar o grau de fiabilidade é necessário tornar mais complexo o algoritmo, o que pode não ser computacionalmente viável. Embora com algumas limitações, existem já soluções comerciais viáveis, para diversas áreas, por exemplo, no caso da videovigilância, no caso dos videojogos, entre outros.

Neste âmbito, uma área que tem sido descorada e só recentemente tem recebido alguma da atenção que merece é o reconhecimento da mão humana, algo bastante complexo, mas que oferece um infinito de possibilidades tecnológicas, outrora longe de serem desenvolvidas.

De facto, a possibilidade de controlar um equipamento com o gesto da mão, causa inúmeras problemáticas em termos tecnológicos, mas deixa em aberto um sem fim de aplicações práticas, como por exemplo ligar e desligar a televisão, ou abrir e fechar uma porta. No entanto se pensarmos mais profundamente, estamos a falar de um interface, com o mesmo tipo de funcionalidade que por exemplo o rato do computador, ou o movimento de toque num ecrã tátil. Temos um interface natural/orgânico, que quem sabe, talvez seja o próximo grande sucesso, talvez até igual ao ecrã tátil.

No momento em que decidi desenvolver este projeto, não havia ainda no mercado o sensor *kinect v2*, que hoje já permite, ao nível do *SDK (Software Development Kit)* detetar e programar alguns movimentos da mão Humana.

O sensor que adquiri foi o *kinect v1 (for windows)* que não possui tais funcionalidades, pelo menos não ao nível das bibliotecas de desenvolvimento disponibilizadas pelo fabricante.

Mesmo existindo uma nova versão, é importante referir, que devido ao facto da versão um do sensor ter sido comercializada durante quatro anos (desde 2010) e devido ao facto da versão dois ter um preço elevado, existem, sem sombra de dúvida, muitos utilizadores e programadores no Mundo, que ainda usam a versão um, até porque o *hardware* é semelhante.

A meu ver, torna-se assim importante, trazer este funcionalismo para a primeira versão e também para outros dispositivos semelhantes, que existem no mercado.

No âmbito deste trabalho, proponho-me a desenvolver, então, um protótipo que permita demonstrar que é possível usar o movimento da mão humana como interface natural, recorrendo à versão do sensor *kinect v1 for Windows*.

Para executar tal protótipo, irei usar a mais recente versão do *SDK*, a v1.8 e a linguagem de programação *C#*, bem como a ferramenta de *Microsoft Visual Studio* 2013, que me foi posta à disposição, através do protocolo da Universidade, com a Microsoft, o programa *Dreamspark*.

Capítulo I

Visão Computacional

1.1. Problemática

Para mergulharmos no ramo da visão computacional temos que perceber que uma imagem captada por um dispositivo, por exemplo uma máquina fotográfica, é apenas uma representação a duas dimensões de um espaço tridimensional. A imagem captada representa um ambiente com determinados objetos, que podem ser os mesmos mas terem imagens diferentes, por exemplo através da iluminação a que o objeto está exposto, ou da sua sombra, ou da cor. Por exemplo, a sombra ajuda-nos a perceber que um determinado objeto é tridimensional, e quando olhamos para uma imagem (bidimensional) interpretamos esse objeto como tendo profundidade, embora na realidade isso não seja verdade, a imagem que estamos a ver é apenas bidimensional. A sombra pode assim representar uma dificuldade acrescida quando estamos a analisar uma imagem.

1.1.1. Luz, Reflexão e Sombra

Segundo Forsyth e Ponce (2011), a iluminação depende da intensidade da luz e da geometria. Mas a intensidade em si pode variar não só consoante a fonte da mesma, mas também com outros fatores como por exemplo a direção da mesma ou a reflexão por parte de outras superfícies. Segundo os autores, a reflexão da radiação por parte de outros elementos tem o nome de *diffuse reflection* e pode ela mesma ser dependente de outros fatores, como por exemplo a direção da radiação ou a matéria do objeto.

A sombra (que obviamente também depende da iluminação) gera formas complexas, derivadas da geometria do objeto e da fonte de radiação e que possuem fronteiras ténues. Embora se possa pensar que na análise computacional de uma imagem a sombra possa dificultar (facto que pode acontecer). Esta pode também ajudar a inferir a informação face à exposição do objeto/ambiente.

1.1.2. Cor

De acordo com os autores acima referenciados, a sensibilidade de captação, quer do olho humano, quer dos equipamentos de captação de imagens é bastante variável. A cor captada varia ainda com os outros fatores já descritos (por exemplo a iluminação). No entanto, existem algoritmos que permitem corrigir ou melhorar uma imagem, mesmo quando esta é captada num ambiente que distorce a sua real cor.

Este procedimento de correção torna-se fundamental quando tentamos, por exemplo, comparar imagens. Ainda que uma determinada imagem seja a captação de um ambiente igual, a influência de fatores como a iluminação, altera a cor captada. (Forsyth e Ponce, 2011)

1.2. Análise do Movimento

A análise do movimento através da visão computacional tem sido alvo de desenvolvimento ao longo dos últimos anos, nomeadamente na componente de análise dos movimentos do corpo humano, sendo esta uma das áreas de maior enfoque da visão computacional.

Os avanços tecnológicos das últimas décadas têm catalisado a pesquisa neste domínio, sendo atualmente possível um sistema de captura, transferência e processamento em tempo real. No entanto, segundo Biasi (2004), de modo geral, a captura de movimento é um problema prático bastante complexo e que leva a dificuldades teóricas para as quais não existe frequentemente uma solução estável, robusta ou computacionalmente viável.

Não é de estranhar o interesse pela análise do movimento, atendendo a que, do ponto de vista técnico, o tema é rico e desafiante, uma vez que implica a segmentação e análise de estruturas que muitas vezes sofrem alteração em termos de topologia e

geralmente envolvem movimento de tipo não rígido com oclusão parcial ou até total (Gavrila, 1999) (Pinho, et al, 2004).

A análise do movimento, nomeadamente do corpo humano, tem vindo a ser motivada pelo objetivo de melhorar a interação homem/máquina nas diversas aplicações da tecnologia. Não é de estranhar, pois o movimento faz parte da nossa forma de comunicar e interagir, sendo que a maior parte da pesquisa desenvolvida no domínio da análise de movimento humano tem tido incidência no seguimento dos sujeitos e inferência da sua pose. (Moeslind, 2001).

Importa também referir que, o domínio da análise de movimento, recorrendo à visão computacional, comporta já hoje, um vasto outro leque de aplicações práticas como por exemplo, a análise do tráfego automóvel, a previsão da condição atmosférica com base na movimentação das nuvens, ou a análise da deformação de materiais, (Pinho, et al, 2004).

Estudos psicológicos indicam que a perceção humana do movimento é um mecanismo de agrupamento visual primário, enquanto a capacidade de detetar/inferir objetos a partir de caraterísticas bidimensionais (ex: cor, textura, forma) é desenvolvida posteriormente (Spelke, 1994).

1.3. Classificação do Movimento

Segundo Tavares (2000), citado por Pinho (2004), existem dois tipos de movimentos: os rígidos e os não rígidos. O movimento rígido entende que a distância entre quaisquer dois pontos do objeto é preservada (não existe deformação), o objeto não estica nem dobra, sendo mantida a sua curvatura média e curvatura Gaussiana da superfície.

Quanto ao movimento não rígido, este pode ser classificado de diversas formas. Segundo a abordagem de Wang (2003), citado Pinho (2004), o movimento não rígido pode ser articulado, elástico ou fluido. O movimento articulado acontece caso as partes rígidas de um corpo se movimentem independentemente das restantes, imaginemos o

braço de um robot, por exemplo. O movimento elástico caracteriza-se pela continuidade/suavidade do movimento não rígido, em que existe algum grau de deformação, por exemplo o acenar de um lenço. O movimento fluido é o movimento não rígido que não satisfaz a restrição de continuidade, podendo envolver variações na sua topologia, bem como deformações turbulentas.

1.4. Captura do Movimento

Embora existam bastantes aplicações de análise de movimento com inúmeras técnicas de captura subjacentes, podemos considerar, de forma geral, que os tipos de imagens utilizadas são gerados através da combinação de uma fonte de iluminação e da reflexão e absorção de energia por parte dos corpos presentes na cena. (Gonzalez, 2003)

Segundo Reynolds, (2002), citado por Pinho (2004), a referência a “cena” e a “iluminação” pressupõe não só as fontes de luz visíveis ao olho humano, mas outras como por exemplo: de origem eletromagnética (radar, infravermelho, ou raio-x), ultrassons, etc. De modo geral, os elementos da cena, podem não ser objetos familiares, mas sim moléculas, rochas, ou outros, que dependendo da fonte de luz, refletem ou transmitem a energia.

Por exemplo, a monitorização de insetos, pode incluir dispositivos óticos (câmaras) e optoelectrónicos, como identificação por radiofrequência, rádio telemetria, entre outros, podendo as várias técnicas ser usadas para capturar e analisar os vários movimentos.

1.5. Sistema de Processamento de Imagem

A estrutura de um sistema típico de processamento de imagens, de forma holística, começa pela captação, tarefa executada por parte de um sensor físico que capta a energia radiada pelo objeto que se pretende visualizar. De seguida, os dados são “digitalizados”, ou seja, convertidos num formato digital e posteriormente tratados por um *software* que executa tarefas específicas, sendo a exibição dos dados, tipicamente feita num monitor. (Gonzalez, 2003)

Independentemente do tipo de sistema em causa, podem tipicamente ser identificados três aspetos comuns, nomeadamente:

1. É necessário segmentar o objeto em causa, do resto da imagem;
2. Seguidamente, as várias áreas da imagem podem, ou não, ser transformadas noutro tipo de representação por forma a diminuir a quantidade de informação (por exemplo, depois de isolar o objeto pretendido, tratar a restante cena, como uma cor sólida);
3. Por fim, criar uma definição de como o objeto em causa irá ser seguido pelos vários "quadros", ou seja, ao longo da sequência de imagens. (Pinho, 2004)

1.6. Pré-processamento

Após a captura da imagem, poderá ser benéfico o uso de algumas técnicas de pré-processamento de imagem, no sentido de facilitar a posterior análise dos dados. Para tal efeito, poderão ser utilizadas técnicas de realce, restauração, de processamento ou de compressão de imagens. (Gonzalez, 2003)

No que toca às técnicas de realce, estas têm o objetivo de fazer sobressair detalhes escondidos, ou salientar determinadas características de interesse na imagem (por exemplo aumentando o contraste).

A restauração da imagem, normalmente incorpora modelos probabilísticos ou matemáticos, com vista a melhorar a aparência da imagem.

O processamento de imagem atualmente é feito através da cor, uma vez que esta permite a extração de características da imagem, em vez da escala de cinza.

A compressão de imagens consiste na utilização de técnicas de redução de espaço de armazenamento e por conseguinte redução dos tempos de comunicação das mesmas.

1.7. Segmentação

A segmentação tem por objetivo determinar as regiões da imagem (cena) que correspondem aos objetos em causa.

A segmentação de imagens, baseada em modelos, consiste na exploração de informações prévias acerca dos objetos e/ou da cena. Tais informações são extraídas com a imposição de restrições, por exemplo sobre as fronteiras das regiões a segmentar.

A segmentação pode ser realizada com recurso a vários tipos de dados. No entanto, as principais abordagens têm em vista o uso de dados temporais e espaciais.

1.7.1. Temporal

Existindo a hipótese de, tanto o fundo da cena como o sensor (câmara), serem estáticos, pode ser usado como forma de inferir movimento a comparação entre imagens (diferença entre as mesmas), por forma a obter o movimento de um dado objeto. A determinação da referida diferença, pode ser concretizada, através da subtração entre imagens ou pelo cálculo do fluxo ótico. (Moeslund, 2001)

1.7.1.1. Subtração

A subtração de imagens é normalmente utilizada e carateriza-se pelo fato de subtrair a intensidade (ou gradiente) de cada *pixel* entre várias imagens, sendo o resultado o movimento (e o ruído) entre imagens (a não ser que o objeto tenha a mesma cor de fundo da cena).

Para otimizar este método, pode ser adquirida uma imagem de fundo sem ruído e sem qualquer tipo de objeto e usar a mesma como referência na subtração, por forma a aumentar o grau de sensibilidade da análise.

O método de subtração apresenta bons resultados em ambientes controlados, mas apresenta várias limitações em situações com determinado grau de variação. (Correia, 1995)

Uma versão mais avançada deste método consiste em atualizar a imagem de fundo ao longo do processamento, uma vez que esta pode sofrer alterações. Por exemplo: uma câmara no exterior está suscetível a alterações climatéricas que alteram o fundo da cena (luz, chuva, entre outros). (Haritaoglu, 1998)

1.7.1.2. Fluxo Ótico

O método do fluxo ótico baseia-se no movimento coerente de pontos ou caraterísticas entre imagens, por forma a obter a segmentação. (Correia, 2001)

O fluxo ótico representa a distribuição 2D da velocidade aparente do movimento, sob os padrões de fundo da imagem. (Franco e Boyer, citados por Simas et al, 2007).

Consiste, portanto num campo denso de velocidade, onde cada pixel, no plano da imagem, tem associado um único vetor de velocidade. (Barbosa et al, citado por Simas et al, 2007).

Existem várias abordagens para a segmentação com recurso ao fluxo ótico, por exemplo:

- Segundo Yamamoto (1991), citado por Pinho (2004), os parâmetros do movimento de uma parte do corpo humano podem ser estimados a partir do fluxo ótico dos vários pontos que a constituem, por comparação ao movimento de um modelo pré-definido;

- Segundo Bregler (1997) e Bregler (1998), citado por Pinho (2004), cada pixel pode ser representado pelo seu fluxo ótico, sendo os mesmos agrupados em objetos binários (designados por *blobs*) com movimentos coerentes e representados por uma combinação de *Gaussianas* multivariadas;

- Por fim, segundo Glu (1994), citado por Pinho (2004), as orlas de intensidade em imagens consecutivas podem ser segmentadas pelo seu comprimento e contraste, usando fluxo ótico.

- Para um exemplo mais prático, podemos olhar para a seguinte fórmula, que refere que a deteção de movimento, após a segmentação do *background* (elementos de fundo) e *foreground* (movimento), é representada através da seguinte equação (que traduz o movimento):

$$u = \frac{dx}{dt}, v = \frac{dy}{dt}$$

Fig. 1 Equação do fluxo ótico

(Fonte: Britto, s/d, pp. 9)

Em que *dt* representa o objeto em movimento ao longo do tempo e *dx*, *dy* o seu deslocamento.

Quando a segmentação se baseia em dados temporais (todos os exemplos acima), é assumido que o único objeto que se move na cena é o objeto de estudo e portanto que as alterações dos pixéis representam movimento. Embora esta abordagem apresente as limitações já referidas, apresenta-se como uma alternativa mais fácil ao estudo de dados espaciais.

1.7.2. Espacial

A segmentação através de dados espaciais pode ser realizada através de binarização ou segundo abordagens estatísticas.

1.7.2.1. Binarização

A segmentação por binarização consiste num processo baseado em hipóteses referentes à cena ou ao objeto em causa. Por outro lado, a segmentação por abordagens estatísticas pressupõe a exploração de algumas hipóteses de aparência com base nos métodos de subtração.

Caso a cor ou intensidade (escala de cinza) do objeto possa ser distinguida do fundo (cena), pode-se então segmentar imagens por binarização. (Darrell, 1994 e Iwai, 1999).

Outra hipótese de binarização consiste em usar marcas passivas ou ativas no objeto em causa, que possibilitem a fácil segmentação como por exemplo cores brilhantes e díspares de todas as outras em cena (Campbell, 1995 e Gonçalves, 1998) ou então recorrendo a um emissor e sensor de infravermelhos e a marcas especiais refletoras no objeto em causa (Iwasawa, 1997).

1.7.2.2. Abordagens Estatísticas

As abordagens estatísticas são variadas e não seguem um método exato. Os autores que as implementam, de alguma forma, procuram adaptar abordagens matemáticas para resolver o problema em mãos, não seguindo uma metodologia exata.
De um modo geral as abordagens estatísticas usam as características individuais de cada pixel, ou de grupos de pixéis, por exemplo, a cor ou o contorno, para extrair o objeto da imagem.

Algumas destas abordagens, por forma a detetar um objeto, consideram uma sequência de imagens de fundo e calculam a média e variância da intensidade/cor de cada pixel (ou grupo) ao longo do tempo. Seguidamente, em cada imagem o pixel é comparado com as estatísticas da imagem de fundo e classificado como pertencendo ou não à mesma. Segue-se a questão da deteção do movimento, que é resolvida neste método, fazendo a subtração entre os quadros/imagens e modelando a mesma numa mistura de duas distribuições *Gaussianas*. Este sistema é bastante rápido, no entanto, não pode ser implementado quando a cena tem orlas de intensidade muito perto dos objetos. (Wang, 2003)

Stauffer (1999), exemplificado na figura abaixo, usa uma mistura de *Gaussianas* adaptativas para modelar o fundo da cena e posteriormente classifica cada pixel, como

pertencendo ou não ao fundo, com base na distribuição *Gaussiana* que melhor o representa, tendo este método a capacidade de atuar em meios reais (variáveis).

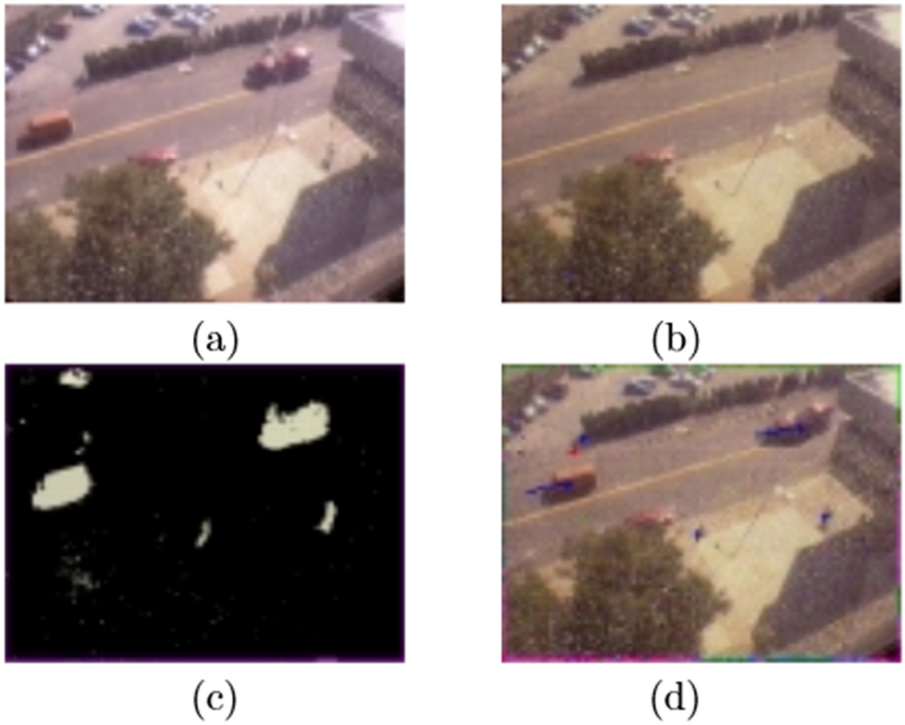

Fig. 2 Resultados da segmentação apresentada por Stauffer

Legenda:

a) imagem inicial;

b) imagem composta pelas médias das Gaussianas mais prováveis no modelo do fundo da cena;

c) segmentação com realce aos pixels dos objetos a seguir;

d) imagem inicial com a informação de seguimento sobreposta (a azul). (Fonte: Stauffer, 1999)

Cuchiara (2000), também recorre a uma abordagem estatística para segmentar de forma rápida objetos em movimento em ambientes não controlados (neste caso circuitos de videovigilância).

Wren (1997), analisa o movimento humano de um sujeito que é modelado com base num conjunto de *blobs* com cores e estatísticas espaciais individuais e cada pixel é classificado como pertencendo a um dos *blobs* de acordo com as suas propriedades espaciais e de cor.

Segundo Kass (1988) e Cootes (1992), citados por Pinho (2004), outra possível abordagem estatística consiste em utilizar contornos estáticos ou dinâmicos. Os contornos estáticos referem-se ao uso pré-definido de estruturas estáticas que representam a borda ou contorno do objeto, tais como segmentos. Os contornos dinâmicos, ou ativos, são utilizados de forma idêntica aos estáticos, com a diferença de se poderem ajustar aos objetos e geralmente funcionarem quando a sua estrutura é desconhecida.

1.7.2.2.1. Contornos Ativos

Os contornos dinâmicos ou ativos, têm vindo a receber crescente atenção ao longo dos anos, uma vez que a sua flexibilidade e versatilidade para a modelação e representação de objetos, destaca este método dos demais. (Mcinerney, 1996) (Jain, 1998)

“Um modelo deformável é ativo no sentido de ser capaz de se adaptar aos dados, conciliando as restrições geométricas da forma com as evidências locais da imagem.” (Pinho 2004, pp. 6)

A temática, na área de modelos deformáveis, pode ser subdividida entre duas classes: os modelos de forma livre ou os modelos deformáveis paramétricos.

Os modelos de forma livre, também chamados de *snakes*, ou de modelos de contornos ativos, representam quaisquer formas arbitrárias desde que obedeçam a uma qualquer restrição de regularidade, como por exemplo a continuidade ou suavidade.

Os modelos deformáveis paramétricos usam uma forma paramétrica, ou um protótipo com os seus modos de deformação. Tais modelos são normalmente utilizados quando existe informação prévia sobre a forma geométrica do objeto e é possível representar a mesma por um conjunto reduzido de parâmetros. (Jain, 1998)

O exemplo do uso de *snakes* pode ser encontrado em (Hanek, 2004), onde as imagens são segmentadas com uso no algoritmo de contração da densidade da curva (*Contracting Curve Density*), sendo o algoritmo empregue para seguir movimentos de estruturas ao longo de várias imagens, usando uma função de aproximação entre os dados representados e a curva modelo, posteriormente recorre a modelos suavizados para determinar a densidade posterior, sendo a curva contraída no sentido de se tornar um contorno do objeto (curva simples), exemplificado na figura abaixo.

Fig. 3 Resultados da segmentação, apresentada por Hanek

De realçar a correção do erro (otimização da curva) que ocorre de iteração para iteração. (Fonte: Hanek, 2004)

Capítulo II

Deteção da Mão Humana, Estudo da Abordagem de Davison

A autoria do seguinte algoritmo é do Professor Doutor Andrew Davison, que leciona no Departamento de Engenharia Computacional da Faculdade de Engenharia *Prince of Songkla University (PSU)*, em *Hat Yai*, Tailândia.

Esta solução consegue identificar e extrair o contorno da mão humana através do contraste e posteriormente identificar os seus dedos, sem recurso a sensores de infravermelhos ou ultrassons, usando para tal o processo de segmentação espacial, denominado Binarização, por forma conseguir encontrar o *blob* (nuvem de pontos que compõe a mão).

Posteriormente são encontrados os contornos (extremos do *blob*) e os seus defeitos (espaços entre os dedos), permitindo assim identificar os dedos através da sua posição no *blob*.

2.1. Tecnologia

Os recursos necessários para demonstrar na prática esta solução são os seguintes:

- Meio de captação de imagens e processamento de imagens (*pc* e *webcam*);
- Possivelmente uma luva para aumentar o contraste entre a mão e o fundo;
- *Software*, nomeadamente as bibliotecas *OpenCV* e *JavaCV*.

2.1.1. OpenCV

OpenCV (*Open Source Computer Vision Library*), é uma livraria de *software* para visão e aprendizagem computacional. O seu principal objetivo é proporcionar uma infraestrutura comum para o desenvolvimento de aplicações nesta área, bem como acelerar o desenvolvimento das mesmas, abstraindo os programadores das tarefas

repetitivas de baixo nível, bem como proporcionando um vasto leque de algoritmos de visão e aprendizagem computacional (cerca de 2500).

De entre a vasta panóplia de algoritmos que a biblioteca possui podemos salientar os seguintes:

- Deteção e reconhecimento de faces humanas;
- Identificação de objetos;
- Classificação de ações humanas em vídeo;
- Identificar e seguir movimentos em vídeo;
- Produzir nuvens de pontos a partir de captações de vídeo e extrair modelos 3D de objetos/ambientes através de matrizes de mapeamento;
- Agregar várias imagens por forma a criar uma imagem HD de um cenário (por exemplo: panorama);
- Introduzir camadas de realidade aumentada;

A comunidade *OpenCV* abrange 47 mil pessoas e é vastamente utilizada por grandes empresas de renome mundial, como a Google, Yahoo, Microsoft, IBM, entre outras, bem como grupos de pesquisa e até entidades governamentais. (OpenCV, 2014)

2.1.2. JavaCV

JavaCV é uma biblioteca *wrapper* que permite o interface com outras bibliotecas escritas em outras linguagens. *JavaCV* faz o mapeamento das funções de bibliotecas fonte para as suas próprias classes, o que permite o seu uso de forma transparente ao programar em Java.

JavaCV suporta outras livrarias além de *OpenCV*, como por exemplo:

- *FFmpeg;*
- *Libdc1394;*
- *PGR FlyCapture;*
- *OpenKinect;*
- *VideoInput;*
- *ARToolKitPlus.*

De destacar na lista acima a biblioteca *OpenKinect*, que se trata de uma biblioteca *open source* para o sensor *Kinect for Windows* da Microsoft. Este surgiu depois da Microsoft disponibilizar o *Software Development Kit* (SDK) do sensor *Kinect* e criar uma versão para programadores designada *Kinect for Windows*.

Por forma a melhor a experiência de desenvolver sob esta livraria, existem ainda uma série de classes (as *utility classes*), que tornam mais fácil e rápido o desenvolvimento, suportando por exemplo:

- Aceleração por *hardware* de imagens (*CanvasFrame* e *GLCanvasFrame);*
- Código para utilização e otimização de processamento *multi-core* (*Parallel);*
- Fácil calibração geométrica e de cores para câmaras/projetores *(GeometricCalibrator, ProCamGeometricCalibrator, ProCamColorCalibrator);*
- Deteção de pontos com propriedades específicas, os *feature points* (*Object Finder*);
- Conjunto de classes que permitem o alinhamento direto de sistemas projetor/câmara, em que o projetor emite e a câmara analisa (*GNImageAligner, ProjectiveTransformer, ProjectiveColorTransformer, ProCamTransformer, and ReflectanceInitializer*);
- Deteção de *blobs* na imagem, ou seja regiões da imagem que tem propriedades específicas ou que estas variam num determinado intervalo, por exemplo de luminosidade (*blobs*).

Esta biblioteca possui ainda uma série de classes compatíveis com o sistema operativo a*ndroid*, possibilitando assim o desenvolvimento para plataformas que utilizem este sistema operativo, nomeadamente equipamentos móveis, *tablets* e até portáteis ou os mais recentes *smart wear gear*, equipamentos como por exemplo relógios que são “vestidos” pelo utilizador. (JavaCV, 2014)

2.2. Identificação da Mão (*blob*)

A imagem captada corresponde não só á imagem da mão mas também ao ambiente que a rodeia, assim é necessário definir corretamente a zona de trabalho, identificando a localização e região da imagem que contém a mão (*blob*).

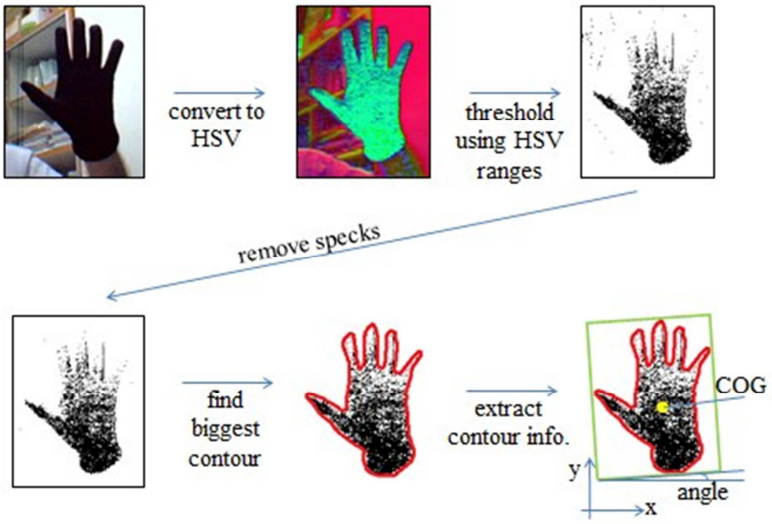

Fig. 4 Processo de identificação da mão, apresentado por Davison

(Fonte: Davison, s/d)

Esta solução faz então a conversão da imagem para o espectro HSV (*Hue, Saturation, Value*), ou Tonalidade, Pureza (reflexão) e Brilho, e estabelece um limite (*threshold*) por comparação aos dados de HSV que são de forma prévia e estática configurada no algoritmo.

Posteriormente são removidos os dados que não têm interesse e encontrado o contorno máximo da nuvem de pontos que compõe o *blob*. Esta informação é extraída, a fim de encontrar o "centro de gravide", o COG (*Center Of Gravity*).

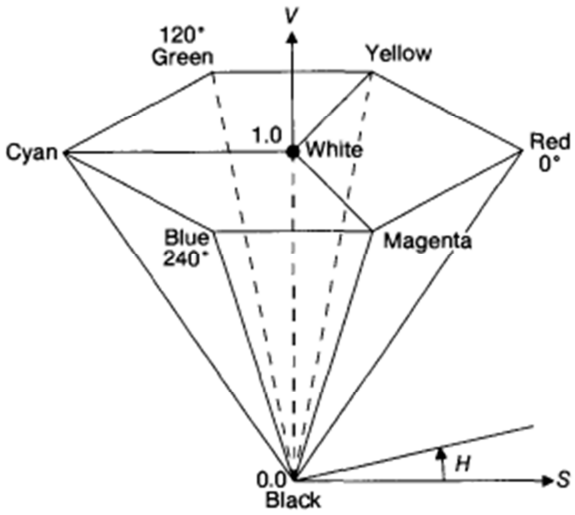

Fig. 5 Espectro HSV

(Fonte: Foley, 1990, pp. 590)

O autor (Davison), define previamente e de forma estática os valores de HSV, que irão ser comparados com os pixéis da imagem por forma a encontrar o *blob*.

2.3. Estabelecer o Contorno

Depois de encontrada a mão, esta não passa de uma nuvem de pontos, sendo então necessário aplicar duas fórmulas matemáticas para a obtenção de uma forma aproximada à forma da mão presente na imagem. São elas a *Convex Hull*, ou Fecho Convexo e através de uma série de pontos máximos e mínimos encontrar os defeitos na Convexa (espaços entre os dedos).

Segundo Goodrich (2002), de uma forma prática, a *Convex Hull*, em Português, Fecho Convexo, ou Convexa, de um conjunto de pontos, é semelhante a atar um elástico ao longo do perímetro máximo que compõe os pontos e encolher o mesmo até um ponto de equilíbrio.

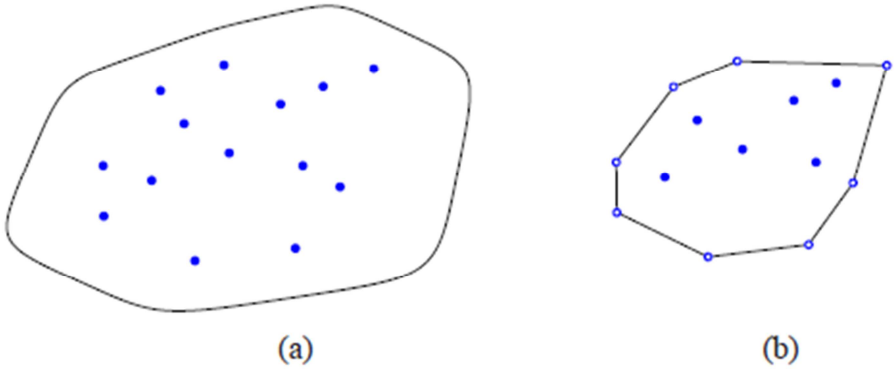

Fig. 6 Fecho Convexo

(Fonte: Goodrich, 2002, pp. 572)

Para encontrar os defeitos do fecho convexo, ou espaços sem pontos (*Convexity Defects*), que representam os espaços entre os dedos é necessário executar três sub-tarefas, sendo elas:

- Encontrar os pontos referentes as pontas dos dedos;
- Encontrar os pontos de referência entre os dedos;
- Identificar cada dedo, consoante a posição que ocupa na mão (exemplo: polegar).

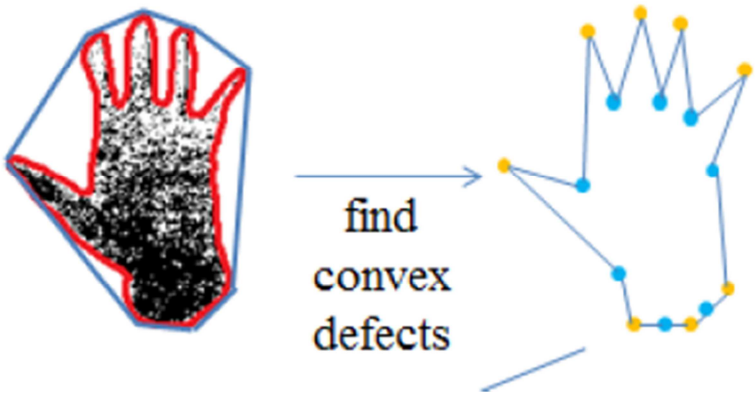

Fig. 7 Encontrar os defeitos do fecho convexo

(Fonte: Davison, s/d)

3.4. Resultados

De seguida são apresentados sobre a forma de imagens os resultados desta abordagem, executados pelo autor.

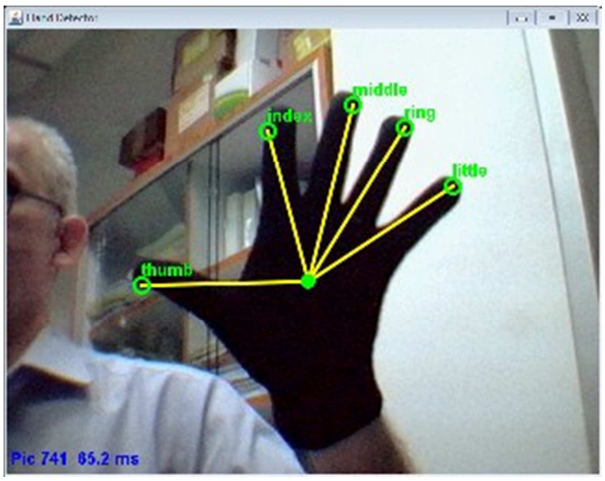

Fig. 8 Resultado com sucesso - Davison

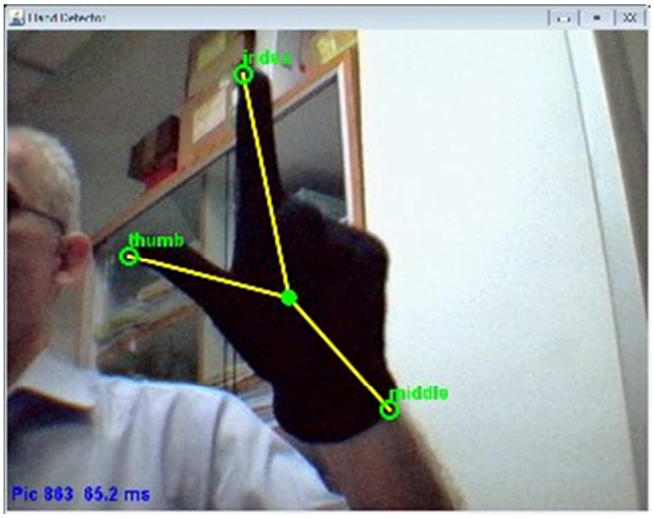

Fig. 9 Resultado com erro - Davison

(Fonte: Davison, s/d)

De seguida são apresentados os resultados que consegui, resultantes do meu teste a esta abordagem.

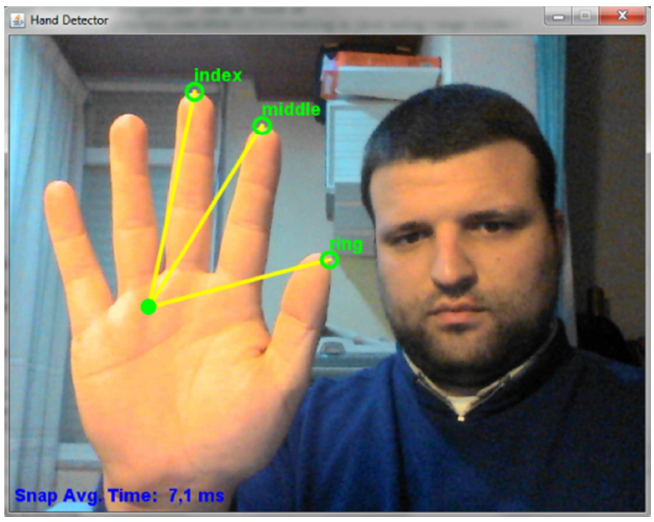

Fig. 10 Resultado com erro 1

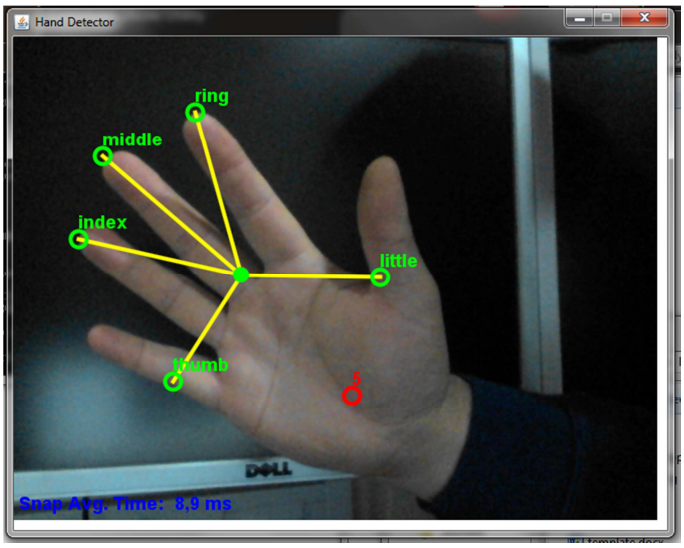

Fig. 11 Resultado com erro 2

Neste último resultado existe um bom nível de deteção, mas ainda assim é reconhecida a mão contrária, uma vez que o código não faz o tratamento a fim de detetar qual das mãos está a ser usada.

2.5. Análise de Resultados

Depois de analisado o exemplo, rapidamente se conclui que existem limitações à solução proposta, nomeadamente:

- É impossível identificar a mão fechada (porque não existem defeitos na *Convex Hull*);
- A anatomia da mão humana permite que o polegar toque qualquer outro dedo da mão. Tal movimento não é interpretado, uma vez que o espaço interior é considerado fechado (imagem abaixo);
- Tem de existir forte contraste entre o *foreground* (mão) e o *background* (fundo), mesmo usando uma luva, por vezes é impossível a deteção;

No entanto, apesar de todas estas limitações, podemos concluir que se trata de resolução possível para esta temática (identificação da mão) e não obstante das limitações apresentadas, a solução tem algumas vantagens. Por exemplo:

- É possível identificar e seguir corretamente os dedos ainda que o ângulo entre os mesmos varie;
- É uma solução pouco complexa e facilmente computável (rápida);
- Não necessita de outras tecnologias (infravermelhos, ultrassons, etc).

As imagens abaixo ilustram melhor o tipo de movimentos conseguidos e as limitações:

(Fonte: Davison, 2013)

Fig. 12 Resultados Possíveis – Davison

Fig. 13 Resultados Impossíveis - Davison

Capítulo III

Deteção da Mão Humana com Sensor Kinect v1, Protótipo de um Interface Natural

3.1. Natural User Interface

NUI (*Natural User Interface*) é um acrónimo frequentemente usado na área da programação ou do design. Trata-se de uma camada abstrata que permite e facilita a interação Homem-Máquina. O termo natural, neste contexto, significa que se aproxima à maneira de interagir do Humano, é mais orgânico e mais intuitivo e inteligível pelo utilizador.

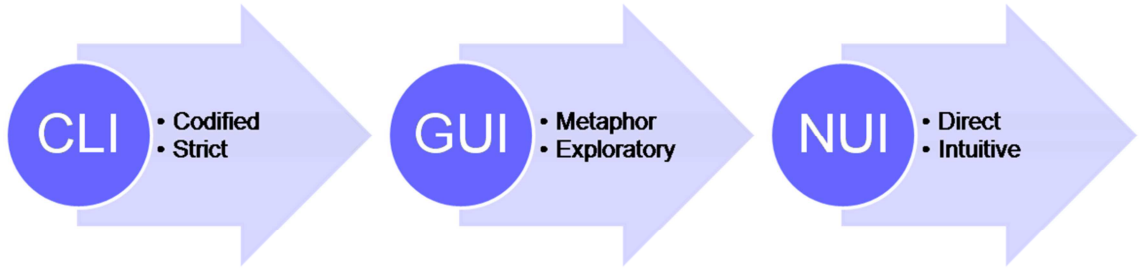

Fig. 14 Evolução dos Interfaces

Na imagem podemos ver aquilo que se considera a evolução dos interfaces e verificar que os mesmos estão cada vez mais orientados para um interface orgânico/natural, capaz de aproximar, cada vez mais a tecnologia ao Humano.

(Wikipédia, Natural User Interface, s/d)

Se olharmos para a história recente da tecnologia e nos colocarmos no ponto onde nos encontramos hoje é fácil perceber que não vamos ficar por aqui. Existe um caminho a seguir, uma evolução, na questão dos interfaces e é fácil perceber para onde caminhamos.

"The keyboard and mouse are going to become obsolete just like the pencil did."

(Hodges, 2012)

3.2. Tecnologia

De seguida é detalhada a tecnologia usada no projeto, o dispositivo *kinect* e o *software* usado, nomeadamente as linguagens de programação e ferramentas de desenvolvimento utilizadas.

3.2.1. Sensor Kinect

Fig. 15 Sensor Kinect versão 1

O nome *Kinect* refere-se à conhecida linha de sensores de movimento da Microsoft. O equipamento existe sob a forma de duas versões com aplicações finais distintas, mas funcionamentos idênticos: o sensor para a consola de jogos *Xbox* e o sensor para aplicações comerciais/programadores (*Kinect for Windows*). Existem à data duas versões de ambos, a versão um e a versão dois do dispositivo e cada versão do referido *hardware* é acompanhado de um *Software Development Kit* (*SDK*) específico.

Para o efeito da demonstração em causa neste relatório, é contemplada a versão um do *hardware* (e forçosamente também do *SDK*), isto porque, à data da aquisição do equipamento era a única existente, tendo a segunda versão sido disponibilizada no mercado mais recentemente.

A versão *Kinect v2* integra já no *SDK v2,* funcionalidades de reconhecimento da mão Humana e, embora ainda não consiga identificar os dedos ou movimentos mais complexos, consegue, por exemplo identificar a mão aberta e mão fechada.

Embora possa parecer algo limitada, esta funcionalidade já permite, um nível de interação semelhante, por exemplo, à do rato do computador, pois temos dois tipos de movimentos (mão aberta e fechada, pode ser visto como botão esquerdo e direito) e o movimento da mão (o movimento do rato do pc).

Ora, este nível de funcionalismo já permite interagir com o pc, para as mais diversas tarefas, como navegar na internet, desenhar, etc.

No entanto esta função é algo recente e não foi implementada na primeira versão do *hardware/sdk,* não existindo portanto na versão *v1*. (Wikipédia, Kinect, s/d)

3.2.1.1. Modo de Funcionamento

Na sua anatomia, o sensor é composto de, uma câmara normal semelhante a uma *webcam*, capaz de produzir imagens a 1600x1200 *pixéis* de resolução, dois microfones (considerado um *array*), uma luz de infravermelhos (que irradia o cenário) e um sensor de infravermelhos (que capta a informação de profundidade) e produz uma imagem de

profundidade (falsa cor, por exemplo escala de cinza) de 640x480 *pixéis* de resolução, com uma precisão na ordem de um centímetro (de erro máximo possível) a dois metros de distância.

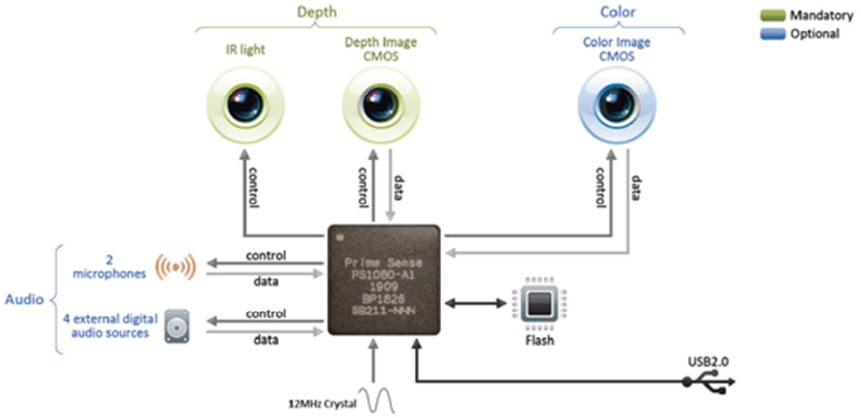

Fig. 16 Componentes do Sensor Kinect

O modo de funcionamento do mapa de profundidade, produzido pelo emissor e pelo sensor de infravermelhos é o seguinte:

Quando o emissor de infravermelhos irradia um cenário, a reflexão varia consoante a distância a que determinado corpo se encontra (exemplificado na imagem abaixo), produzindo assim uma mapa de profundidade.

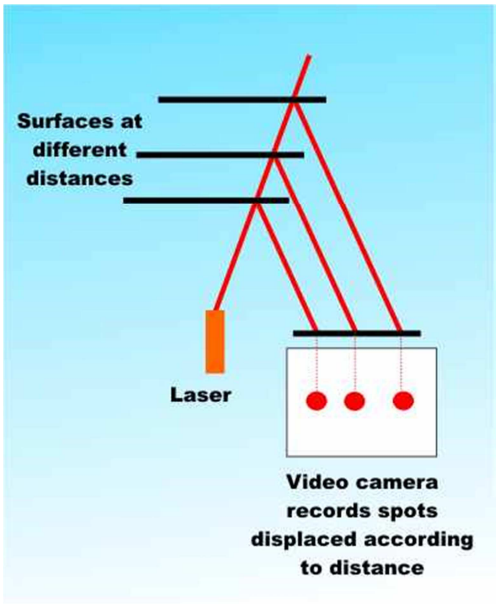

Fig. 17 Captação da profundidade por infravermelhos

O mapa de profundidade, por sua vez, é uma camada de informação adicional (uma terceira dimensão) à imagem captada pela camara (2D), produzindo assim uma imagem mais rica, no sentido em que tem mais informação, do que a imagem captada por uma câmara normal. *(Fairhead, s/d)*

Esta camada adicional pode então ser usada para extrair profundidade e identificar corpos presentes no cenário, muito à semelhança da abordagem de Davison com o contraste.

3.2.2. Software Development Kits

As versões dos *SDKs* da Microsoft são bastantes e existem atualmente 4 versões principais para fins distintos:

- *Kinect for Xbox 360* (para a consola referida);
- *Kinect for Windows v1* (para aplicações comerciais com o *hardware v1*);
- *Kinect for Xbox One* (para a referida consola);

- *Kinect for Windows v2* (para a versão *v2* do *hardware*).

Para os efeitos deste estudo apenas irei estudar a versão *Kinect for Windows v1* do *hardware* e *SDK*.

Para a versão que está em causa (*v1)*, foram ainda disponibilizadas pela Microsoft, duas versões experimentais iniciais, a versão beta 1 e beta 2. Estas versões são atualmente obsoletas, mas ainda é possível encontrar muitos exemplos de aplicações desenvolvidas com recurso a esses *SDKs* pela internet. Infelizmente existem questões de incompatibilidade entre as versões *beta1*, *beta2* e *v1.0* (e posteriores) do *SDK*, o que faz com que todos esses exemplos deixem de estar funcionais e atualizados, pois as alterações são de fundo, nomeadamente ao nível dos nomes e forma de trabalhar das principais classes. De referir que a versão mais atual, à data, é o *SDK v1*.8 e desconhece-se se a Microsoft tem intenção de prolongar o desenvolvimento da versão 1 tendo já um novo *hardware* no mercado (a *v2*).

O *SDK Kinect* oficial suporta as seguintes linguagens de programação:

- ***C++***, e as suas variantes da Microsoft, nomeadamente o ***Managed C++*** e o ***C++/CLI****;*
- ***C#*** (ou *C Sharp*), a linguagem desenvolvida pela própria Microsoft;
- ***Visual Basic***, uma linguagem também da Microsoft voltada para o ambiente gráfico;

Por fim, é ainda importante referir, de forma muito resumida, que todas estas linguagens podem funcionar sobre a plataforma *dotNET* (ou *Framework .NET*) da Microsoft, o que permite a compilação para um código intermédio que é executado sobre uma máquina virtual (*virtual machine*). Permitindo assim que esta camada virtual (*vm*) atue como um *middleware* e permita o uso de várias linguagens compatíveis com *.NET* na mesma aplicação/solução.

(Microsoft, Learn the basics, s/d)

Existem ainda, outros *SDKs* não oficiais que suportam o sensor *Kinect*, nomeadamente:

- ***Libfreenect***, um *SDK Open Source*, da *OpenKinect*. Pode ser utilizado nos mais variados sistemas operativos, como *Linux, Windows* e *Mac* e permite programar nas mais variadas linguagens como:
    - *Python, C, C++, ActionScript, C#, Java JNI, Java JNA, Javascript; Common Lisp, GLib.*

(OpenKinect, About, s/d)

- ***OpenNI SDK***, faz parte do *OpenNI Framework* e é desenvolvido pela *OpenNI* (*Open Natural Interaction*), uma organização sem fins lucrativos que tem por objetivo, catalisar a interoperabilidade da tecnologia que usar interfaces naturais ou orgânicos. O *OpenNI Framework* integra uma série de *APIs* (*Application Programming Interface*) de suporte á área dos interfaces naturais, como por exemplo os gestos da mão. O *SDK é open source* e pela forma como está implementado funciona com outros dispositivos que não só o *Kinect*, além disso funciona com as mesmas linguagem que o *libfreenect* da *OpenKinect*.

(OpenNI, About OpenNI, s/d)

- ***Cinder Framework***, apesar de não ser um *SDK*, é um *framework open source* que permite integrar o *SDK* oficial da Microsoft com o *OpenNI* e programar em *C++*. Suporta os sistemas operativos *OSX* e *iOS* e também *Windows*;

(Cinder, About, s/d)

Para os efeitos deste projeto, como já havia sido referido, irá ser utilizado o SDK do próprio fabricante, na sua versão 1.8.

3.2.2.1. SDK Microsoft Kinect for Windows v1.8

A deteção e seguimento do esqueleto é a funcionalidade principal do sensor e por conseguinte está implementada ao nível das bibliotecas de desenvolvimento. A Microsoft tem procurado tirar o máximo partido deste interface natural (*NUI – Natural User Interface*) para aplicar na sua consola de jogos, a *Xbox 360*, criando assim um segmento de jogos para a sua consola, que se distinguem dos demais, pois podem usar o sensor como interface dos movimentos do jogador.

O sensor através do SDK da Microsoft consegue reconhecer 20 pontos de interesse (chamados *Joints*) no esqueleto Humano e consegue com eles modelar um esqueleto virtual que acompanha os movimentos do utilizador, ou somente tratar pontos específicos, sem ter de tratar todo o esqueleto.

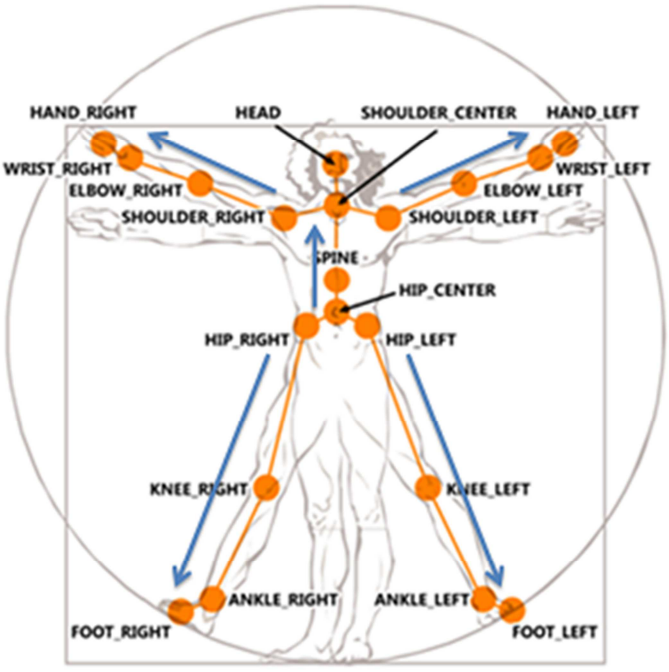

Fig. 18 As 20 articulações do esqueleto detetadas pelo sensor

Para ativar e utilizar esta funcionalidade, o programador (em *C#*), tem de fazer uma chamada ao método “SkeletonStream.Enable” e posteriormente consegue aceder à propriedade “KinectSensor.SkeletonStream” da classe “KinectSensor”, mas não sem antes esperar até que o sensor tenha recolhido amostras suficientes para conseguir apresentar resultados.

Este tipo de espera é tratado com recurso a eventos, nomeadamente os “KinectSensor.SkeletonFrameReady”, “KinectSensor.AllFramesReady”, que tratam especificamente só o esqueleto ou todas as outras *frames* (ou camadas), como a camada de profundidade, ou a da câmara normal.

(Microsoft, Tracking Users with Kinect Skeletal Tracking, s/d)

Esta funcionalidade é ainda mais complexa, no sentido em que consegue identificar e segmentar vários sugeitos no cenário e aconpanhar os movimentos dos mesmos.

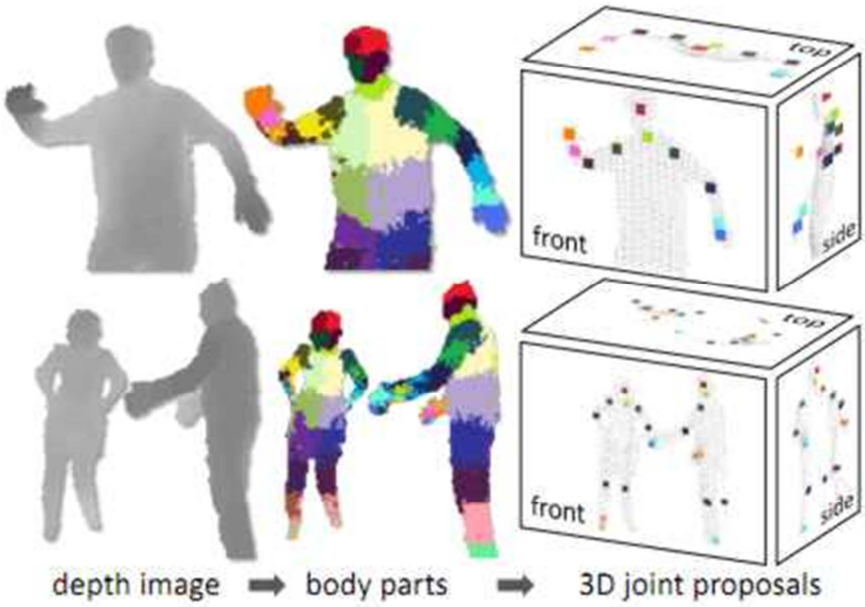

Fig. 19 Deteção de esqueletos de múltiplos indivíduos

(Fairhead, s/d)

Além do esqueleto (*SkeletonFrame*), existem ainda 3 outras *frames* ou camadas que podem ser usadas pelo programador:

a) A camada de cor (*ColorFrame*), captada pela câmara de alta definição do sensor;
b) A camada de profundidade (*DepthFrame*) captada pelo sensor de infravermelhos;
c) E por fim uma camada de agregação (*AllFrames*) que integra a informação de todas as outras e permite de forma síncrona tratar camada-a-camada a informação que está a ser recebida.

As capacidades de desenvolvimento são inúmeras, quer ao nível da indústria dos videojogos, como a vários outros níveis. A possibilidade de utilizar um interface natural para interagir com a tecnologia é realmente um marco da história da Humanidade.

3.2.3. C#

A linguagem de programação C# (C Sharp) foi desenhada e desenvolvida pela Microsoft e foi lançada no ano 2000. É uma linguagem de programação multi-paradigma, uma vez que pode ser imperativa (orientada a procedimentos, sub-rotinas ou funções), funcional (baseada em expressões e funções matemáticas) e orientada a objetos (baseada no conceito de objetos que são estruturas de dados que contem atributos e métodos).

Foi desenvolvida com os objetivos de ser uma linguagem simples, moderna (podendo ser orientada a objetos) e acima de tudo poder ser uma linguagem de uso generalizado principalmente nos ambientes Microsoft (embora seja compatível com outros sistemas operativos).

É uma linguagem que já foi desenvolvida inicialmente com o intuito de integrar a plataforma *dotNet Framework* da Microsoft, que surgiu na mesma altura, fator que catalisou a adoção da linguagem.

Neste momento encontra-se na versão *C# 5* lançada a 15 de Agosto de 2012, sendo que esta versão veio trazer múltiplas funcionalidades de ponta, como métodos assíncronos que permitem acelerar o feedback do programa ao utilizador, nomeadamente em rotinas onde é necessário interagir com recursos que podem bloquear ou tornar mais lento o programa, por exemplo acesso a um ficheiro, ou recurso web. Existe ainda uma versão experimental da funcionalidade *compiler-as-a-service (codename: Roslyn)*, que permite disponibilizar compiladores *self-hosting*, para as linguagens C# e VB.NET, que integram análise semântica, lexical (*parser de tokens*), e permite compilação código intermédio (CIL - *Common Intermediate Language*), por forma a funcionar dinamicamente sobre plataformas virtuais como o *dotNet*. *(Wikipédia, C Sharp Programming Language, s/d)*

O termo *Visual C #*, muitas vezes referido, quando se trata desta linguagem, é na realidade uma implementação da Microsoft à especificação *C#*, que contempla a integração da linguagem na plataforma *.NET Framework* e no *IDE (Integrated Development Environment) Microsoft Visual Studio*. Na maior parte das vezes quando falamos em C# estamos na realidade a falar em Visual C#.

(Microsoft, Visual C#, s/d)

3.2.4. dotNet Framework

O *framework dotNet*, ou *.NET*, é uma livraria de classes, apelidada de FCL (Framework Class Library) desenvolvida pela Microsoft, que tem o objetivo primário de permitir a interoperabilidade de várias linguagens de programação. Os programas escritos para a *framework*, são compilados para um código intermédio CIL (*Common Intermediate Language)* e executados em ambiente de *software* (em vez de hardware), denominado CLR (*Common Language Runtime*). Na prática, o código é executado sobre uma camada virtual (*application virtual machine*) que se traduz num *middleware* entre o *hardware* e o código compilado e que além da interoperabilidade entre linguagens, proporciona outras funcionalidades, nomeadamente gestão de memória, e

tratamento de erros, entre outras. O conjunto das bibliotecas (FCL) e da *virtual machine* (CLR) constituem o *framework dotNet*.

As livrarias de classes *dotNet*, incluem ainda funcionalidades de apoio ao programador, por forma a facilitar, através de classes pré-desenvolvidas, um vasto número tarefas comuns ao programador, como sejam:

- Interface com utilizador;
- Acesso a dados;
- Conectividade com motores de base de dados;
- Criptografia;
- Desenvolvimento Web;
- Algoritmos numéricos;
- Comunicações em rede.

O *framework*, que se tornou bastante popular, tem vindo a sofrer vastas melhorias por parte da Microsoft e vai, à data, na versão 4.5.2. lançada a 5 de Maio do presente ano. Tendo as principais melhorias assentado na área do ASP.NET (linguagem web da Microsoft), nomeadamente com as tarefas de segundo plano assíncronas (á semelhança do métodos assíncronos do C#) e melhorias na inspeção dos *headers http*.

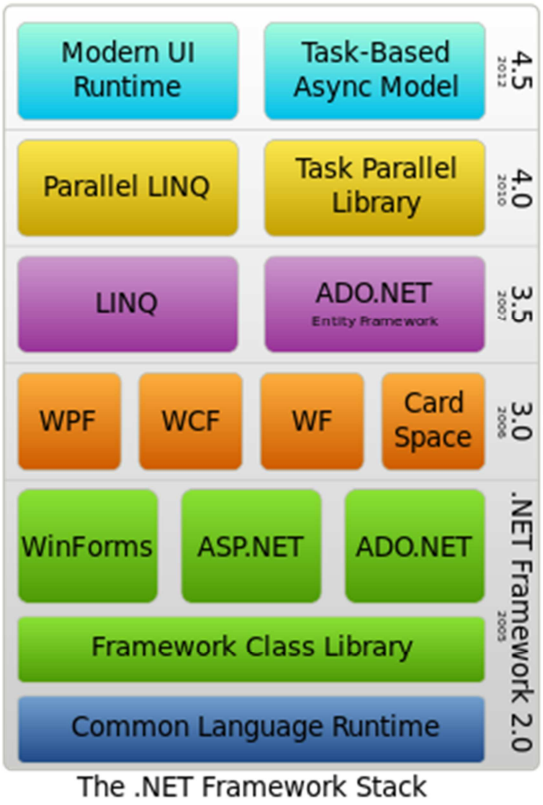

Fig. 20 Linguagens/Funcionalidades do histórico dotNetFramework

Podemos ver na imagem acima apresentada, aquele que foi o histórico de funcionalidades/linguagem que foram sendo adicionadas ao *framework* ao longo das suas versões.

(Wikipédia, .Net Framework Version History, s/d)

3.2.5. Visual Studio

O *Microsoft Visual Studio* é um IDE (*Integrated Development Environment*), ou seja uma aplicação de software que tem por objetivo apoiar o programador no desenvolvimento do software.

Um IDE no geral integra funcionalidades de compilador de código, interpretador de código ou ambos. Inclui ainda funcionalidades de depuração (*debugger*), e automatização da compilação. Além disso proporciona apoios gráficos ao programador,

nomeadamente estabelecendo um interface mais amigável com recurso a funcionalidades gráficas como explorador de classes, explorador de objetos, hierarquias de classes, etc. Os IDEs mais atuais também estão orientados para permitir um desenvolvimento gráfico mais amigável, proporcionando uma vista de programação gráfica que se aproxima do resultado final, por exemplo no desenvolvimento de um *website* ou aplicação gráfica é possível definir os tamanhos e propriedades dos formulários sem editar código.

(Wikipédia, Integrated Development Environment, s/d)

O *Visual Studio* assume-se como a ferramenta de referência, no ambiente de desenvolvimento da Microsoft, pois inclui todas as suas tecnologias e ainda outras linguagens de programação desenvolvidas por terceiros. É capaz de desenvolver não só aplicações em código nativo e em código gerido para o sistema operativo *Windows*, mas também *websites*, *web applications* e *web services*.

Neste momento o portefólio de tecnologias do *Visual Studio* é o seguinte:

- Windows API;
- Windows Forms;
- Windows Presentation Foundation;
- Windows Store
- Microsoft Silverlight

O portefólio de linguagens suportadas é o seguinte:

- C, C++, Managed Extensions for C++ e C++/CLI (Visual C++);
- VB.NET (Visual Basic .NET);
- C# (Visual C#);
- F# (apenas Visual Studio 2010);
- M, Python e Ruby;
- XML/XSLT, HTML/XHTML, Javascript e CSS.

Inclui ainda outras funcionalidades, nomeadamente ao nível do editor de código, inclui suporte a *IntelliSense* (funcionalidade de *autocomplete* e verificação da sintaxe do código) e *code refactoring* (restruturação do código existente sem alterar o seu comportamento externo) e ao nível do depurador permite um funcionamento quer ao nível do código fonte (*source-level debugger*) como ao nível da máquina (*machine-level debugger*). Inclui ainda funcionalidades ao nível do desenho de aplicações gráficas, de *webdesigner*, *class designer*, e criação de estruturas de base de dados (*database schema designer*). Suporta, ainda, outros *plugins* nomeadamente para o controlo de versões do código e *software development lifecycle* (por exemplo com o *Team Foundation Server*).

Neste momento, a versão mais recente deste sofware é a 2013, sendo que existe já uma série de versões *preview* da versão 2014 (ao todo 4 versões CTP – *Community Technology Preview*), tendo a primeira sido lançada a 3 de Junho de 2014 e a última a 6 de Outubro de 2014.

(Wikipédia, Microsoft Visual Studio, s/d)

No âmbito deste projeto a versão utilizada é a versão estável mais atual, ou seja, a versão *Visual Studio 2013*, que me foi colocada à disposição, de forma gratuita, através do protocolo entre a Universidade Atlântica e o programa Microsoft *Dreamspark*.

3.2.5. WPF e XAML

Windows Presentation Foundation (WPF) é um subsistema gráfico para *rendering* de interfaces gráficos de aplicações Windows. Este sistema usa DirectX (API da Microsoft para tratar multimédia), para efetuar o *rendering* gráfico. Esta tecnologia procura separar a lógica do programa da parte do interface e para tal usa uma linguagem baseada no XML para estruturar o ambiente gráfico, do código "*core*" do programa, código esse que é uma linguagem completamente independente.

(Wikipédia, Windows Presentation Foundation, s/d)

XAML (*Extensible Application Markup Language*) é uma linguagem declarativa basseada em XML (*Extensible Markup Language*) e desenvolvida pela Microsoft. É utilizada para inicializar valores estruturados e objetos. Em conjunto com a tecnologia WPF proporcionam ao programador um maior grau de controlo sobre o código, uma vez que o ambiente gráfico está mais desligado da componente lógica do programa e da linguagem em que esta assenta.

(Wikipédia, Extensible Application Markup Language, s/d)

Este projeto faz uso da linguagem XAML, e da tecnologia WPF para apresentar a parte gráfica do programa, sendo posteriormente a lógica do programa, desenvolvida em C#.

3.3. Aplicação Prática

Todo o código que de seguida é apresentado está disponível no anexo 1. O código é composto por dois ficheiros *MainWindow.xaml* que contém a componente gráfica e *MainWindows.xaml.cs* que contém o código em C# que irei agora analisar.

O código deste projeto encontra-se no segundo volume deste trabalho, encadernado à parte, segundo as regras da universidade.

3.3.1. Inicialização do Sensor

O primeiro passo é a inicialização do sensor. É necessário verificar se existe algum sensor disponível, ligado ao computador e posteriormente ativar as funcionalidades pretendidas do mesmo, ao mesmo tempo é necessário realizar o tratamento de erros com *try/catch*. Neste caso, o sensor é declarado (*KinectSensor sensor*) e são ativas das funcionalidades do esqueleto ao nível do *SDK*, com o *sensor.SkeletonStream.Enable()* e do sensor de profundidade, com o *sensor.DepthStream.Enable()* e posteriormente inicializado o sensor com a chamada ao *sensor.Start()*.

Na parte final desta parte do código, importa ainda referir que é colocado um gestor de eventos na chamada ao *AllFramesReady* (que como já falamos é um agregador de todos os *frames* com o maior sincronismo possível). Tal é necessário por forma a controlar o processamento das *frames*, só quando existem *samples* para tratar é que o método respetivo (*sensor_AllFramesReady(…)*) corre o seu código.

3.3.2. Processamento

No método *sensor_AllFramesReady(object, AllFramesReadyEventArgs)* é onde é feito o tratamento das *frames* e todas as chamadas aos métodos de apoio, sendo esta a parte principal do programa.

As *frames* de profundidade e esqueleto são abertas e tratadas (*OpenDepthImageFrame()* e *OpenSkeletonFrame()*) nomeadamente no sentido de encontrar as zonas da imagem que correspondem às mãos e fazer o *crop* dessa imagem para enviar para o *isMakingAFist()*, por forma a verificar se está a mão aberta ou fechada e atualizar a parte gráfica com o rebordo vermelho na imagem da mão.

3.3.3. Métodos de Apoio

Existem ainda vários métodos que desempenham uma função específica no código e importa referir:

short[] convertDepthFrame(short[])

Converte um *array* de *shorts* com informação da imagem em *16bits*, numa imagem a *32bits*, mantendo a informação de índex do utilizador. É comum usar este método e encontrei o mesmo em vários exemplos de código, isto porque a imagem em escala de cinza que o *frame* de profundidade produz é *16bits* e a *frame* de cor é 32bits.

Point getDisplayPosition(DepthImageFrame, Joint)

Recebe um *frame* de profundidade e um *Joint* (articulação), posteriormente converte o *frame* e encontra o ponto x,y (*Point*) onde se encontra a articulação e faz retorno desse valor.

Color PixelColor(ImageSource, int, int)

Recebe uma imagem (*ImageSource*) e dois inteiros, que são dois valores de um ponto *(x,y)* e encontra a cor nesse ponto. Posteriormente faz o retorno da cor no formato *RGB*.

bool isMakingAFist(ImageSource)

Recebe uma imagem da mão (*ImageSource*) e verifica se esta está fechada ou aberta. Analisa a imagem como uma matriz, de 10 em 10 linhas e coluna a coluna. Retorna um valor booleano.

De referir que o algoritmo é muito sensível e causa alguns erros, nomeadamente quando a imagem que é passada (*ImageSource*), não tem a mão bem centrada ou está demasiado afastada (ocupa pouco espaço na janela), ou o movimento é demasiado rápido.

3.3.4. Recursos Usados

Na programação deste protótipo, foram consultados vários exemplos de código sobre a área em questão, sendo que além dos exemplos da Microsoft, que são integrados no *SDK*, importa referir os seguintes:

- O exemplo de código de *tracking* do esqueleto funcional, para a versão v1.0+ do SDK, intitulado *Kinect Skeleton Viewer* de Miles (2012);
- O exemplo de código de *tracking* da mão para a versão *beta1* do SDK, intitulado *Simple Hand Tracking with MS Kinect SDK & WPF* de (Tango, 2011), que embora não funcionasse nem com o referido *SDK*, foi uma boa base de aprendizagem e aproveitamento de código, nomeadamente na parte gráfica e na parte de reconhecimento da mão.

3.3. Resultados

Neste protótipo os resultados são apresentados à semelhança do exemplo de Tango (2011), com uma borda vermelha em redor da imagem de *tracking* de cada mão, a borda vermelha significa punho fechado e é encontrada pelo método *isMakingAFist()* e tornada visível para o utilizador pelo *sensor_AllFramesReady()*.

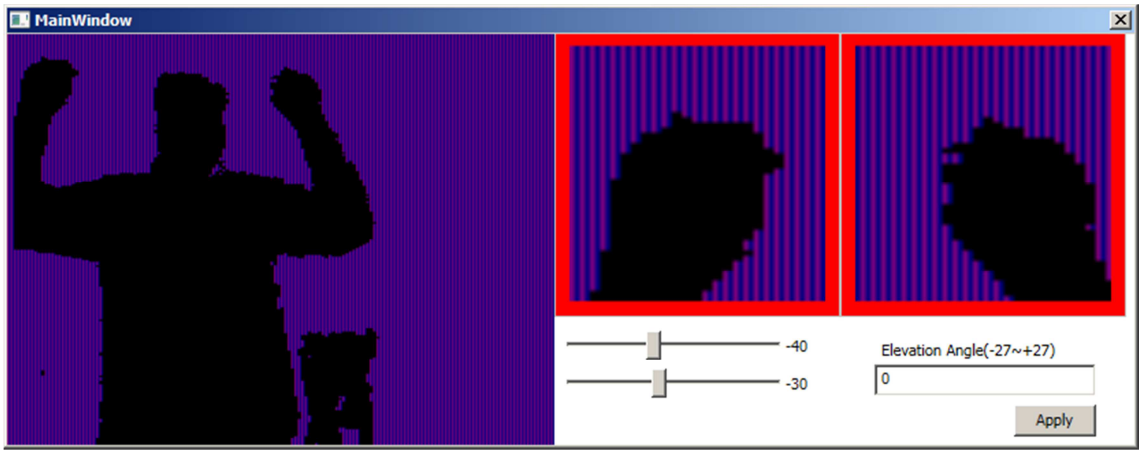

Fig. 21 Deteção com sucesso de ambas as mãos fechadas

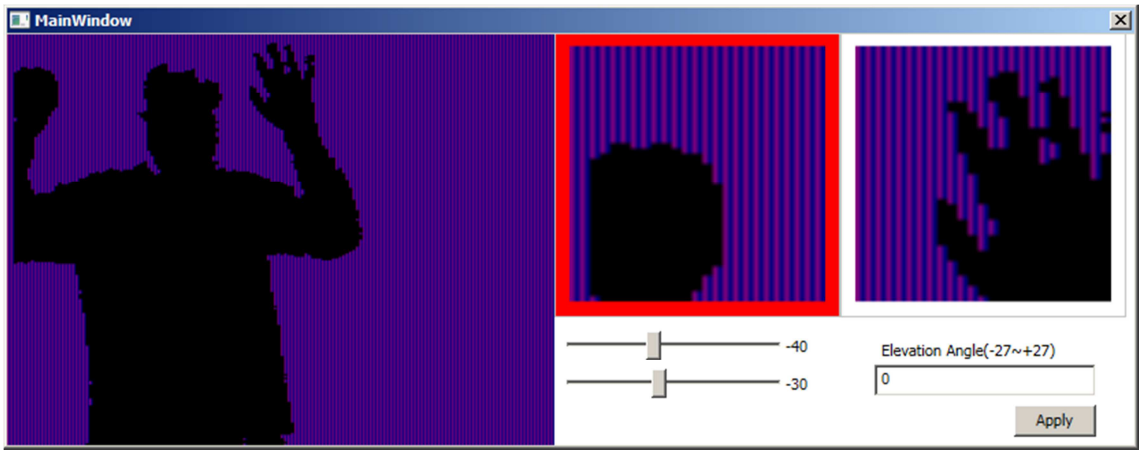

Fig. 22 Deteção com sucesso de uma mão fechada e uma aberta

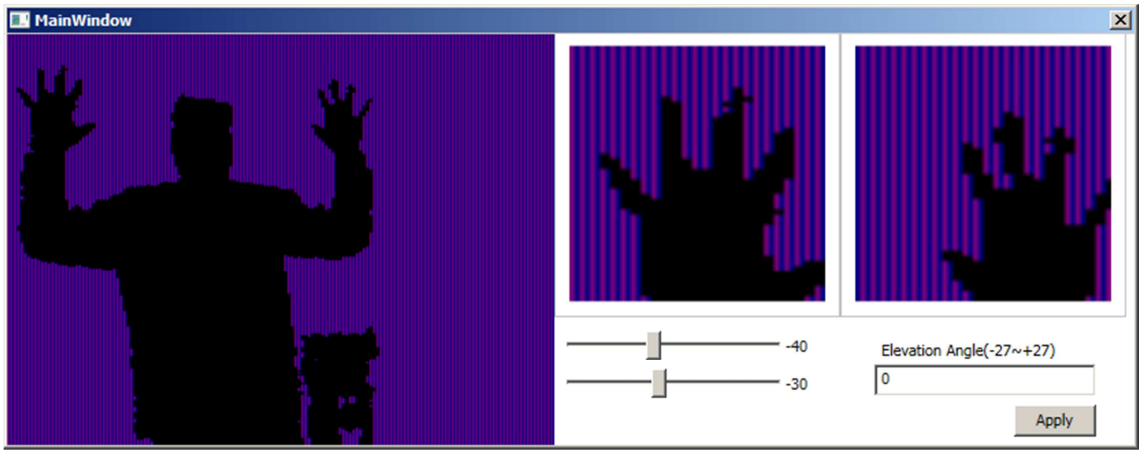

Fig. 23 Deteção com sucesso de ambas as mãos abertas

Apesar dos inúmeros sucessos é ainda possível observar alguns erros de deteção.

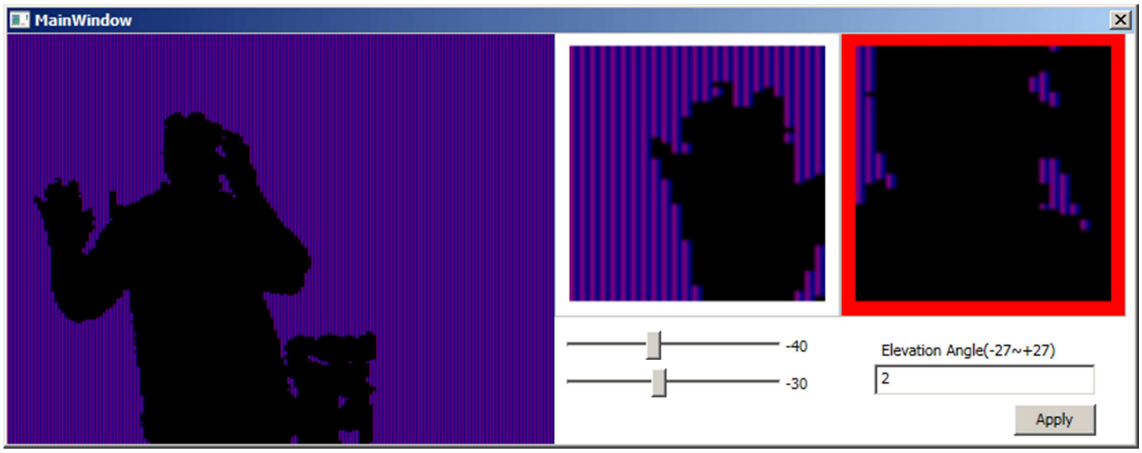

Fig. 24 Erro de deteção por aproximação ao corpo

Nesta imagem acima podemos identificar uma das limitações do programa, como o *isMakingAFist()* se baseia na deteção de pixéis pretos a aproximação ao corpo causa falsos positivos.

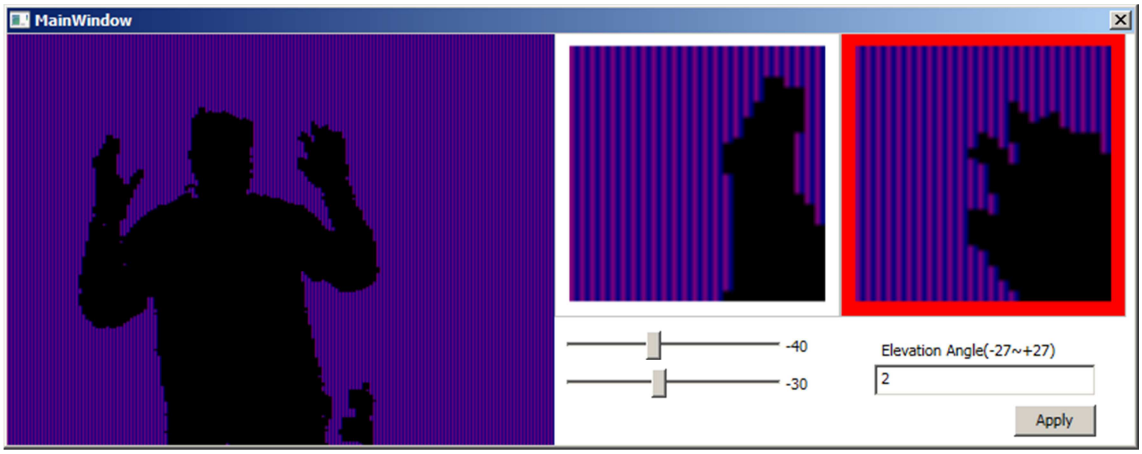

Fig. 25 Erro de deteção pela mão estar entreaberta

Na imagem acima o que acontece é que a mão esquerda está ligeiramente de lado, pode-se ver que ainda assim o programa detetou a mão como estando aberta, ou seja com sucesso. Já na mão direita, pelo facto de estar apenas entreaberta e mal centrada, a mesma apresenta erro.

3.4. Análise de Resultados

Se verificarmos atentamente os resultados, entendemos rapidamente que existe alguma margem de erro. Das experiências realizadas conclui que o erro se deve essencialmente a três problemas, todos eles relacionados com a forma como a mão aberta/mão fechada é detetada, ou seja com o método *isMakingAFist()*.

O primeiro problema é que o código deste método é rudimentar e assenta sob o princípio de que a mão está centrada na imagem que pertence a mão (que foi removida do *frame* principal, pelo seguimento da articulação das mãos). Um algoritmo que possibilitasse maior fiabilidade tornaria o protótipo muito mais sólido, mas infelizmente não consegui encontrar ou desenvolver outro que produzisse melhores resultados.

O segundo problema é a aproximação ao corpo, após vários testes concluí que a aproximação ao corpo da mão cria erro, existe a possibilidade de estar relacionado com

o facto de existir menos contraste entre as profundidades da mão e do cenário, como a mão é seguida como sendo uma articulação (*Joint*) do *frame Skeleton*, todo ele preto, na imagem. No entanto acredito que neste caso específico o problema está novamente no método *isMakingAFist()* pois não segmenta a mão do resto do esqueleto o que faz com que detete o esqueleto (preto) como sendo o punho, ainda que estes estejam em profundidades diferentes.

O terceiro problema tem a ver não só com a forma como é detetado o punho mas com a forma como o próprio kinect funciona. Reparei após vários testes que o programa tende a produzir falsos positivos (mão fechada), aquando do movimento brusco da mão. Penso que isto acontece pela seguinte razão: O *sensor_AllFramesReady*, "alimenta" várias samples do esqueleto e da *frame* de profundidade ao programa, no maior sincronismo possível, no entanto o método *isMakingAFist()*, não tem qualquer tipo de sincronismo, ele trata as *samples* que lhe são passadas num determinado tempo de execução, que penso não seja o mesmo, que aquele a que os resultados são apresentados.

Além destes problemas existe ainda uma limitação desta abordagem e que é clara face à abordagem de Davison. Neste caso, não é possível reconhecer os dedos da mão individualmente ou no conjunto. Embora esta solução seja mais fiável que a de Davison, uma vez que incorpora a informação de profundidade e as funcionalidades de seguimento do esqueleto do *SDK*, é também menos complexa. Se analisarmos o código de deteção da mão de Davison face ao apresentado nesta abordagem rapidamente verificamos que o método *isMakingAFist()* teria de ser amplamente melhorado, senão até completamente refeito para conseguir tais resultados.

Limitações

Encontrei ao longo do desenvolvimento desta pesquisa várias limitações, as quais gostaria de referir, para dar a conhecer ao leitor deste trabalho as problemáticas com que me deparei.

A bibliografia existente é "pesada", de difícil entendimento, a sua leitura é morosa, cansativa e chega a ser desesperante, não só pela complexidade da área, mas essencialmente porque, muitas vezes, os autores expressam as suas ideias, sem introduzir ou explicar antecipadamente os pressupostos.

A grande limitação no desenvolvimento do protótipo foi também a minha capacidade de entender os trâmites da linguagem de programação (*C#)* e das livrarias *kinect*, a sua complexidade e a minha inexperiência causaram muitos fracassos e perdas de tempo. De referir que tive o primeiro contacto com esta tecnologia, com a linguagem e com o sensor *kinect*, aquando do início deste relatório.

O teste da abordagem de Davison também teve os seus constrangimentos, mas desta feita, porque tem de existir um casamento perfeito entre as librarias, *opencv*, *javacv*, entre as versões do sistema operativo (*x86/x64*) e os *frameworks* (*Visual C++)* necessários para a execução do protótipo. De tal forma que a versão mais recente de *javacv* e *opencv* não funcionam, tendo de ser usadas versões específicas, de preferência, num ambiente isolado apenas com o *software* necessário, por exemplo numa máquina virtual ou num computador dedicado para o efeito.

Não podia deixar de referir a questão do tempo. De facto, as limitações em termos de tempo que um aluno do pós-laboral enfrenta, são de facto enormes. Conciliar a vida pessoal, profissional e académica é mais que muitas vezes uma tarefa Herculana.

Por final, o desafio maior e o que acabou por causar mais limitações, foi causado pelos conflitos entre as várias versões do S*DK* da Microsoft. O facto da versão *beta 1, beta 2* e *v1.0+*, terem implementações completamente distintas das diferentes classes,

pois algumas foram extintas, outras renomeadas, outras passaram a ter outros parâmetros, etc. Tal causou um tamanho embaraço ao analisar exemplos de código, muitos da própria Microsoft, o que levou a qua a implementação fosse frustrante e cansativa.

Linhas Futuras

Da análise dos resultados é fácil concluir que o protótipo (*Kinect*) apresenta algumas limitações. Nomeadamente carece de solidez e fiabilidade. A implementação de um algoritmo que permita detetar com maior clareza se a mão está aberta ou fechada torna-se um ponto essencial de desenvolvimento futuro.

Tal algoritmo deve assentar sobre um modelo que não se baseie meramente na procura de pixéis pretos na imagem de profundidade (convertida) mas também de informação de profundidade sobre a própria mão, nomeadamente para possibilitar a deteção com precisão da posição da mão.

Além deste aspeto, é necessário implementar a deteção dos dedos. Eventualmente com uma abordagem semelhante á de Davison no seu algoritmo que usa *OpenCV* e *JavaCV*, ou outro que permita maior fiabilidade. Após este desenvolvimento as hipóteses de estudo crescem exponencialmente, nomeadamente permitindo o reconhecimento de outros gestos e por conseguinte permitindo um interface natural mais rico com a máquina.

Contributos

A presente pesquisa, bem como em grande parte a generalizada da pesquisa na área dos interfaces naturais e da visão computacional tem claros contributos para um sem fim de áreas. Considero que no caso desta pesquisa em concreto, existe um contributo concreto para determinadas áreas, de salientar algumas:

- Na criação de dispositivos de apoio a pessoas com algum grau de deficiência, nomeadamente dispositivos capazes de serem comandados por interface natural;
- Na indústria, nomeadamente no controlo de linhas de produção, a ausência de um equipamento físico para comandar um dispositivo pode trazer inúmeras vantagens;
- Na robótica, através da programação de autómatos com capacidades de entender linguagem natural;
- Em ambientes em que não é possível ou conveniente outro tipo de comunicação. Por exemplo para aplicações militares, debaixo de água, debaixo do solo, na indústria espacial, etc.

Por fim devo referir, que a meu ver, o ponto em esta tecnologia terá o seu auge, será talvez, quando for possível a um computador, perceber os movimentos complexos da linguagem gestual e ser capaz de traduzir a mesma para linguagem escrita. Embora possa parecer um ponto ainda algo distante no tempo, acredito que estamos a caminhar nesse sentido.

Conclusão

A possibilidade um interface natural com a máquina é algo de realmente fascinante. Os novos sensores de infravermelhos, que estão a surgir no mercado, estão a tornar esta tecnologia acessível às massas e catalisar o desenvolvimento da área da Visão Computacional. Embora esta tecnologia tenha sido generalizada no mercado pela indústria dos videojogos, rapidamente surgem outras aplicações práticas que fazem uso dos seus recursos.

Ao longo deste trabalho são demonstradas várias limitações da tecnologia, também é verdade que várias têm vindo a ser ultrapassadas.

O sensor Kinect foi, e é, um marco nesta tecnologia, mas não é o único, existem outros fabricantes que estão a desenvolver ou até já lançaram no mercado soluções semelhantes. Desta concorrência, espera-se um mais rápido desenvolvimento desta área.

No que diz respeito a este trabalho, tenho a concluir que de facto os sensores que integram a tecnologia de infravermelhos e com ela conseguem uma terceira camada (3D) de uma imagem captada em 2D, proporcionam um nível de fiabilidade bastante grande, o que permite aligeirar a complexidade dos algoritmos de deteção e tornar os mesmos computacionalmente viáveis em tempo real e também mais fiáveis.

Especificamente o sensor kinect, ao ser disponibilizado para programadores e aplicações comerciais fora da indústria dos videojogos, veio munir os entusiastas desta área, de um dispositivo acessível e facilmente programável, o que de certa forma o tornou o equipamento de referência, a meu ver, nesta área.

O protótipo apresentado, apesar de algumas falhas, consegue detetar, seguir e reconhecer movimentos da mão, como por exemplo a mão fechada ou aberta, o que, não fosse a margem de erro, poderia proporcionar um interface viável à interação com um computador, por exemplo com o mesmo nível de funcionalidade do rato ou *touchpad*. Embora com as referidas limitações, o protótipo demostra esta teoria.

Bibliografia

Pinho, R.; Tavares, J.; Correia, M. (2004). Introdução à Análise de Movimento usando Visão Computacional, Faculdade de Engenharia da Universidade do Porto; disponível em: http://repositorio-aberto.up.pt/bitstream/10216/171/2/10131.pdf; Último acesso em 10/08/2014;

Gavrila, D.; Davis, L. (1996); 3D Model-Based Tracking of Humans in Action: A multi-view Approach, IEEE Conference on Computer Vision and Pattern Recognition, San Fransisco, USA;

Moeslund, T.; Granum, E. (2001); A Survey of Computer Vision-Based Human Motion Capture, Computer Vision and Image Understanding, pp. 81 e 231 a 268;

Spelke, E.; Vishton, P.; Hofsten, C. (1994). Object Perception, Object-Directed Action, and Physical Knowledge in infancy, pp. 165 a 179;

Tavares, J. (2000); Tese de Doutoramento, Análise de Movimento de Corpos Deformáveis usando Visão Computacional, Faculdade de Engenharia do Porto;

Wang, X; He L.; Wee, W. (2004); Deformable Contour Method: A Constrained Optimization Approach, International Journal of Computer Vision, Volume 59, pp. 87 a 108;

Gonzalvez, J.; Kim, I.; Fua, P.; et al; (2003); Robust Tracking and Segmentation of Human Motion in an Image Sequence, ICASSP 2003 Conference of Acoustics, Speech and Signal Processing, Hong Kong;

Reynolds, D.; Riley, J. (2002); Remote-Sensing, Telemetric and Computer-Based Technologies for Investigation Insect Movement: A survey of Existing and Potential Techniques; Computer and Electronics in Agriculture; pp. 35 e 271 a 307;

Haritaoglu, I.; Harwood, D.; Davis, L.; (1998); W4: Who? When? Where? What? – A Real Time System for Detecting and Tracking People, International conference on Automatic Face and Gesture Recognition, Nara, Japan;

Correia, M. (1995); Dissertação de Mestrado, Análise de Movimento em Sequências de Imagens, Faculdade de Engenharia, Universidade do Porto;

Correia, M. (2001); Tese de Doutoramento, Técnicas Computacionais na Percepção Visual do Movimento, Faculdade de Engenharia, Universidade do Porto;

Bregler, C.; (1997); Learning and Recognizing Human Dynamics in Video Sequences, Proceedings IEEE Computer Vision and Pattern Recognition, San Juan, Puerto Rico;

Bregler, C.; Malik, J.; (1998); Tracking People with Twists and Exponential Maps, International Conference on Computer Vision and Pattern Recognition;

Gu, H.; Shirai, Y.; Asada, M.; (1994); MDL-Based Spatiotemporal Segmentation from Motion in Long Image Sequence, Computer Vision and Pattern Recognition;

Darrell, T.; Maes, P.; Baumberg, B.; et al; (1994); A novel environment for Situated Vision and Behaviour , Workshop for Visual Bahaviours at CVPR-94;

Iwai, Y.; Ogaki, K.; Yacida, M.; (1999); Posture Estimation using Structure and Motion Models, International Conference on Computer Vision, Corfu, Greece;

Campbell, L.; Bobick, A.; (1995); recognition of Human Body Motion Using Phase Space Constraints, International Conference on Computer vision, Cambridge, Massachusetts;

Gonçalves, L.; Bernardo, E.; Perona, P.; (1998); Reach out and touch Space (Motion Learning), International Conference on Automatic Face nad Gesture Recognition, Nara, Japan,

Iwasawa, S.; Ebihara, K.; Ohya, J.; et al; (1997); Real-Time estimation of Human Body Posture from Monocular Thermal Images, Conference on Computer Vision and Pattern Recognition;

Wang, J.; Singh, S.; (2003); Video Analysis of Human Dynamics – A Survey, Real-time Imaging Journal Volume 9 Issue 5, pp. 109 a 129;

Stauffer, C.; Grimson, W.; (1999) Adaptive Background Mixture Models for Real-time Tracking, Computer Vision and Pattern Recognition; pp. 2246 a 2252;

Cuchiara, R.; Grana, C.; Piccardi, M.; et al; (2000); Statistic and Knowledge-based Moving Object detection in Traffic Scenes, Proceedings of the 3rd IEEE Conference on Intelligent Transportation Systems (ITSC2000); Dearbon, Indiana, USA;

Wren, C.; Azarbayejani, A.; Darrell, T.; et al; (1997); Pfinder: Real-Time Tracking of the Human Body, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 19, Issue 7;

Kass, M.; witkin, A.; Terzopoulos, D.; (1988); snake: Active Contour Model, International Journal of Computer Vision, Volume 1, Issue 4; pp. 321 a 331;

Cootes, T.; Taylor, C.; (1992); Active Shape Models: Smart Snakes, British Machine Vision Conference;

Biasi, S.; (2004); Rastreamento de Movimento com Marcadores Óticos Retrorefletores; Disponível online em: http://lvelho.impa.br/ip02/demos/sbiasi-04.pdf; Último acesso em 30/03/2014;

Britto, A.; E Koerich, A.; (s/d); Detecção e Análise de Movimentos em Vídeos; Disponível Online em: http://www.ppgia.pucpr.br/~alekoe/Papers/Seminario2-TVGlobo-Abril2007-color.pdf; Último acesso em 30/03/2014;

Simas, G.; Fickel, G.; Novelo, L.; et al; (2007); Utilizando Visão Computacional para Reconstrução Probabilística 3D e Rastreamento de Movimento; Disponível Online

em: http://www.seer.furg.br/vetor/article/viewFile/1675/817; Último acesso em 29/03/2014;

Franco, J.; Boyer, E; (2005); Fusion of Multi-View Silhouette Cues Using a Space Occupancy Grid. ICCV 05 (10th IEEE International Conference on Computer Vision);

Barbosa, R.; Gallis, R.; Silva, J.; et al; (2005); A Computação do Fluxo Óptico em Imagens Obtidas por um Sistema Móvel de Mapeamento Terrestre. Revista Brasileira de Cartografia;

Hanek, R.; Beetz, M.; (2007) The Contracting Curve Density Algorithm: Fitting Parametric Curve Models to Image using Local Self-Adapting Separation Criteria, International Journal of Computer Vision Volume 59 Issue3; pp. 233 a 258

Jain, A.; Zhong, Y.; Dubuisson-Jolly, M-; (1998); Deformable Template Models: A Review; Signal Processing Volume 71; pp. 109 a 129;

Mcinerney, T.; Terzopoulos, D.; (1996); Deformable Models in Medical Image Analysis: A Survey, Medical Image Analysis; Volume 1, Issue 2;

OpenCV; (2014); About OpenCV; Disponível Online em: http://opencv.org/about.html; Último acesso em: 30/03/2014;

JavaCV; (2014); JavaCV Summary; Disponível Online em: https://code.google.com/p/javacv; Último acesso em 30/03/2014;

Davison, A.; (2013); Vision-based User Interface Programming in Java; Amazon Digital Services, Inc.

Foley, J.; Van Dam, A.; Feiner, S.; et al; (1990); Computer graphics Principles and Practice; Addison-Wesley, 2e edition; pp. 590;

Goodrich, M.; Tamassia, R.; (2002).; Algorithm Design; John Wiley & Sons Inc; pp. 572;

Davison, A.; (data desconhecida); Chapter VBI-6. Hand and Finger Detection; Disponível em: http://fivedots.coe.psu.ac.th/~ad/jg/nui055/index.html; Acedido em 30/03/2014;

Wikipédia; (data desconhecida); Kinect; Disponível online em: http://en.wikipedia.org/wiki/Kinect; Último acesso em 02/08/2014;

Microsoft; (data desconhecida); Learn the basics; Disponível online em: http://www.microsoft.com/en-us/kinectforwindows/develop/learn.aspx; Último acesso em 02/11/2014;

OpenKinect; (s/d); About; Disponível online em: http://openkinect.org/wiki/Main_Page; Último acesso em: 05/08/2014;

Cinder; (s/d); About; Disponível online em http://libcinder.org/about/; Último acesso em 08/11/2014;

OpenNI; (2013); About OpenNI; Disponível online em: http://www.openni.tk/openni/post/2013/08/15/About-OpenNI; Último acesso em: 08/11/2014;

Fairhead, H. (s/d); All About Kinect; Disponível online em: http://www.i-programmer.info/babbages-bag/2003-kinect-the-technology-.html; Último acesso a 08/11/2014;

Microsoft; (s/d); Tracking Users with Kinect Skeletal Tracking; Disponível onlinte em: http://msdn.microsoft.com/en-us/library/jj131025.aspx; Último acesso em 08/11/2014;

Forsyth, D., Ponce, J.; (2011); Computer Vision: A Modern Approach, Segunda Edição; Prentice Hall.

Miles, R.; (2012); Kinect Skeleton Viewer; Disponível Online em: http://www.robmiles.com/journal/2012/2/7/kinect-skeleton-viewer.html; Último acesso em 10/11/2014;

Tango, C. (2011); Simple Hand Tracking with MS Kinect SDK & WPF; Disponível online em: https://kinecthandtracking.codeplex.com/; Último acesso em 10/11/2014;

Wikipédia; (s/d); C Sharp Programming Language; Disponível online em: http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29; Último acesso em 10/11/2014;

Microsoft; (s/d); Visual C#; Disponível online em: http://msdn.microsoft.com/en-us/library/kx37x362.aspx; Último acesso em: 11/11/2014;

Wikipédia; (s/d); Integrated Development Environment; Disponível online em: http://en.wikipedia.org/wiki/Integrated_development_environment; Último acesso em 11/11/2014;

Wikipédia; (s/d); Microsoft Visual Studio; Disponível online em: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio; Último acesso em 11/11/2014;

Wikipédia; (s/d); Windows Presentation Fondation; Disponível online em: http://en.wikipedia.org/wiki/Windows_Presentation_Foundation; Último acesso em 11/11/2014;

Wikipédia; (s/d); Extensible Application Markup Language; Disponível online em: http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language; Último acesso em 11/11/2014;

Microsoft; (s/d); Introduction to WPF; Disponível online em: http://msdn.microsoft.com/en-us/library/aa970268%28v=vs.110%29.aspx; Último acesso em 11/11/2014;

Wikipédia; (s/d); Natural User Interface; Disponível online em: http://en.wikipedia.org/wiki/Natural_user_interface; Último acesso em: 11/11/2014;

Hodges, S. (2012); A world of NUI: Steve Hodges; Disponível online em: http://blogs.microsoft.com/next/2012/01/26/a-world-of-nui-steve-hodges/; Último acesso 11/11/2014.