FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sistema de realidade virtual (Oculus Rift) para controlo remoto de braço robótico

Daniel António Teixeira Varum

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Leonel Caseiro Morgado (*PhD, Habil.*)

Orientador secundário: António Fernando Vasconcelos Cunha Castro Coelho (*PhD*)

Orientador secundário: Luís André Freitas da Rocha (*PhD*)

26 de Junho de 2015

Sistema de realidade virtual (Oculus Rift) para controlo remoto de braço robótico

Daniel António Teixeira Varum

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Luís Filipe Pinto de Almeida Teixeira (*PhD*)

Vogal Externo: Hugo Alexandre Paredes Guedes da Silva (*PhD*)

Orientador: Leonel Caseiro Morgado (*PhD, Habil.*)

Orientador secundário: António Fernando Vasconcelos Cunha Castro Coelho (*PhD*)

---

26 de Junho de 2015

Resumo

Recentemente, as interfaces imersivas entraram em voga, notoriamente aliadas a interações naturais. Nesta área, existem os *Head-Mounted Displays* e os sensores de movimento que fazem *tracking* do corpo e reconhecimento de gestos do utilizador. Uma potencial aplicação destas tecnologias é a teleoperação de robôs sem necessidade de experiência técnica.

O intuito é criar uma prova de conceito altamente experimental que alie estas áreas, numa vertente meramente exploratória. Pretende-se, então, desenvolver um sistema para controlar remotamente um braço robótico de forma natural. Para tal, usa-se o dispositivo de realidade virtual Oculus Rift e uma câmara estereoscópica (3D) ou equivalente para captar e ver a perspetiva do robô como se fosse a do utilizador e usa-se ainda o sensor de movimento Leap Motion para emitir comandos gestuais e controlar o robô, fazendo o reconhecimento de gestos e *tracking* dos braços, mãos e dedos do utilizador. Este braço robótico concretiza, depois, tarefas simples com objetos de pequena dimensão. As tarefas consideradas resumem-se a agarrar e pousar objetos, encaixar e desencaixar peças e até mesmo premir botões.

Nas referências analisadas, são descritos sistemas que tentam resolver problemas semelhantes usando tecnologias como sensores inerciais e algoritmos de visão por computador, mas não consideram as questões da precisão e da visualização do espaço onde o robô opera.

A visualização desse espaço com o Oculus Rift foi conseguida de modo fiável, com perceção da profundidade e tridimensionalidade dos objetos, pelo menos para distâncias à câmara inferiores a 2 metros.

O reconhecimento de gestos foi implementado através de restrições aos movimentos detetados pelo Leap Motion, com uma taxa de reconhecimento superior a 90%.

O controlo do braço robótico é realizado maioritariamente por meio de imitação dos movimentos da mão do utilizador. O controlo propriamente dito foi implementado sobre a *framework* ROS e o *software* MoveIt! e mostrou-se bastante intuitivo, dada a correspondência direta entre os movimentos da mão imitada e os da garra robótica. Apesar disso, este sistema é menos preciso do que o controlo por *joysticks* e o desfasamento temporal entre as ordens do utilizador e as ações do robô é problemático e deverá ser examinado no futuro.

Palavras-chave: Interação pessoa-computador, Interação natural, Sensor de movimento, Deteção de gestos, Leap Motion, Imersividade, Realidade virtual, Oculus Rift, Controlo remoto, Braço robótico, Robótica.

Abstract

Recently, immersive interfaces have entered the scene, notoriously coupled with natural interactions. We have Head-Mounted Displays that allow for virtual worlds to be rendered and viewed in 3D, and motion sensors that track the user's body and recognize certain gestures. One of many potential usages of these technologies is the teleoperation of robots without the need of technical expertise.

The idea is to build a highly experimental proof of concept that combines these areas of study. The goal is to develop a system for controlling a robotic arm remotely and in a natural fashion. The virtual reality device called Oculus Rift and an equivalent to a stereoscopic camera are used to capture and view the robot's perspective of the scene. The Leap Motion sensor is used to detect gestural commands and control the robot, by tracking the user's arms, hands and fingers. This robotic arm then executes simple tasks that require interaction with small objects. These tasks are: grab and place objects, attach and detach parts and even press buttons.

The studied references describe systems that try to solve similar problems by using technologies such as inertial sensors and computer vision algorithms, but do not consider the issues of robot movement precision and visualization of the physical space where the robot operates.

The visualization of that space with the Oculus Rift was achieved reliably, attaining perception of depth and tridimensionality of objects, at least for distances to the camera below 2 meters.

The recognition of gestures was implemented through the definition of restrictions to the movements detected by the Leap Motion, with a recognition rate greater than 90%.

The robotic arm control is mainly done by mimicking the user's hand movements. The implementation of the robot control was accomplished using the ROS framework and the MoveIt! software. The results showed this was highly intuitive, given the direct correlation between the user's mimicked hand movements and those of the robotic gripper. However, this solution is less precise than control through joysticks and the delay between the commands given and the actions executed is problematic and should be addressed in the future.

Keywords: Human-computer interaction, Natural interaction, Motion sensor, Gesture detection, Leap Motion, Immersion, Virtual reality, Oculus Rift, Remote control, Robotic arm, Robotics.

Agradecimentos

Aos meus orientadores, agradeço a persistência com que se certificaram de que estava a dar o meu máximo, conhecendo as minhas capacidades.

Ao professor Armando Sousa, agradeço a introdução dada ao *Robot Operating System* e os guias que me ajudaram a compreender a programação de robôs, ao estilo de *crash course*.

Aos meus colegas do InMERSE, agradeço os conhecimentos partilhados e a companhia em vários eventos.

Obrigado aos meus amigos que me apoiaram e animaram sempre que me sentia desmotivado dado os vários obstáculos, por vezes desmedidos, com que me deparei.

Fica ainda muito por dizer, mas prefiro deixar os restantes agradecimentos em pessoa.

Daniel Varum

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

2D	Bidimensional
3D	Tridimensional
API	*Application Programming Interface*
DK	*Development Kit*
EMG	Eletromiografia
FPS	*Frames Per Second*
HCI	*Human-Computer Interaction*
HMD	*Head-Mounted Display*
HMM	*Hidden Markov Model*
HTA	*Hierarchical Task Analysis*
IP	*Internet Protocol*
IV	Infravermelhos
JSON	*JavaScript Object Notation*
NUI	*Natural User Interface*
OLED	*Organic Light-Emitting Diode*
RGB	*Red Green Blue*
RNA	Redes Neuronais Artificiais
ROS	*Robot Operating System*
SDK	*Software Development Kit*
SMO	*Sequential Minimal Optimization*
SVM	*Support Vector Machine*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
WIMP	*Windows, Icons, Menus, Pointers*

Capítulo 1

Introdução

1.1 Enquadramento

As interfaces de *software* mais comuns são puramente bidimensionais e muitas baseiam-se no estilo "*Windows, Icons, Menus, Pointers*", ou WIMP, desde cedo difundido pelos sistemas operativos comerciais. Por vezes, estas interfaces são visualmente complexas e exigem interações pouco intuitivas. Recentemente, as interfaces imersivas entraram em voga, notoriamente aliadas a interações mais naturais. Nesta área, existem sistemas de realidade virtual e aumentada que dão uso a *Head-Mounted Displays*, ou HMDs. Estes aparelhos de telepresença, como o nome indica, são montados na cabeça do utilizador e fornecem uma visão estereoscópica (tridimensional) de um cenário gerado computacionalmente ou de um espaço real mas remoto, criando um nível de imersão espacial já previsto por filmes de ficção científica. Expetavelmente, estes sistemas casam na perfeição com outro tipo de tecnologia aparentemente futurista: sensores de movimento que fazem *tracking* do corpo e reconhecimento de gestos do utilizador. Ultimamente, isto ganhou ímpeto e popularidade devido às últimas gerações de consolas de videojogos, que integraram este tipo de tecnologia. Assim, é possível interagir com a realidade vivida no HMD de uma forma natural, através de gestos e poses do corpo completo ou dos braços e mãos do utilizador.

Outra área, não tão recente mas em contínuo desenvolvimento e com enorme potencial, é robótica. A automação de vários processos – sobretudo industriais –, a realização de tarefas em ambientes inadequados a humanos e tarefas que requerem alta precisão são algumas das suas potencialidades. Porque não conciliar todas estas áreas? Que género de soluções inovadoras poderá daí surgir? Serão essas soluções viáveis com as tecnologias atuais?

Nesta dissertação, faz-se um estudo multidisciplinar que compreende todas estas áreas e constrói-se uma prova de conceito altamente experimental, numa tentativa de responder a estas perguntas. Para tal, assume-se um conjunto de tarefas genéricas com objetos de pequena

dimensão, como agarrar e pousar objetos, encaixar e desencaixar peças e até mesmo premir botões. Procura-se uma forma alternativa de executar estas tarefas, com precisão suficiente e potencialmente à distância, por meio das tecnologias referidas acima.

Este trabalho insere-se no projeto InMERSE entre o INESC TEC e a Portugal Telecom Inovação & Sistemas.

1.2 Motivação e objetivos

Pretende-se desenvolver um sistema para controlar remotamente um braço robótico, usando um dispositivo de realidade virtual (HMD) e uma câmara estereoscópica (3D) ou equivalente para captar e ver a perspetiva do robô, e usando ainda um sensor de movimento para fazer *tracking* do corpo e reconhecimento de gestos do utilizador, permitindo que este emita comandos com os seus braços, mãos e dedos. Este braço robótico concretiza, então, tarefas simples com objetos pequenos.

Primeiramente, tenciona-se resolver o problema da visualização do ambiente onde se encontram os objetos com os quais interagir, de tal forma que se facilite a execução das tarefas. É com este motivo que se propõe usar um HMD e vídeo 3D, pois deverá mitigar dificuldades resultantes da diferença de perspetiva – a perspetiva do utilizador é a do robô.

Pretende-se também resolver o problema da deteção dos movimentos, leitura da posição e reconhecimento de vários gestos e poses do utilizador, nomeadamente dos seus braços, mãos e dedos.

Propõe-se, ainda, atacar o problema do controlo adequado dos movimentos do braço robótico para executar o género de tarefas considerado com a precisão necessária.

O intuito é criar uma prova de conceito que alie estas áreas, numa vertente meramente exploratória. Com isto, procura-se medir a viabilidade das tecnologias referidas quando aplicadas em simultâneo em cenários de teleoperação de robôs, nos quais se executam tarefas simples como as mencionadas na secção anterior. Procura-se ainda verificar até que ponto estas tecnologias conseguem competir com as atualmente utilizadas para controlo remoto, como monitores 2D para visualização e *joysticks* para controlo do robô.

1.3 Estrutura da dissertação

Para além da Introdução, este documento inclui quatro outros capítulos. No **Capítulo 2**, faz-se o estudo das tecnologias de interesse, dos algoritmos a usar e dos trabalhos relacionados. No **Capítulo 3**, faz-se uma descrição detalhada do problema em mãos e da abordagem proposta para o resolver. No **Capítulo 4**, faz-se uma descrição da implementação dos vários componentes da solução e discutem-se os testes efetuados e os resultados obtidos. No **Capítulo 5**, tiram-se conclusões relativas ao estudo e trabalho realizados, discute-se a satisfação dos objetivos e dão-se direções para trabalho futuro.

Capítulo 2

Revisão Bibliográfica

Neste capítulo, faz-se o estudo das tecnologias de interesse, dos algoritmos a usar e dos trabalhos relacionados.

2.1 *Head-Mounted Displays*

2.1.1 Introdução

Um HMD é montado na cabeça do utilizador, com um ou mais monitores em frente aos seus olhos. A ideia é simular a visão humana e criar um ambiente imersivo tridimensional, apresentando uma imagem ligeiramente diferente a cada olho através do *headset* (Bell 2014). O *output* para cada olho deve reproduzir o respetivo campo de visão, como ilustrado na **Figura 1**.

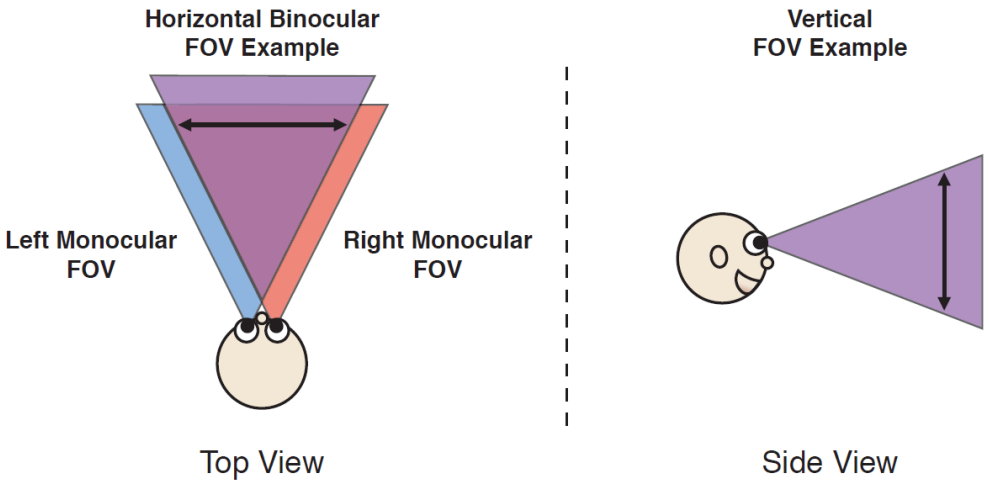

Figura 1: Campos de visão horizontal e vertical (Bell 2014)**.**

HMDs com um campo de visão largo promovem uma maior imersividade, pois apresentam mais informação visual do ambiente (Bell 2014).

No caso de exibição de uma cena 3D gerada computacionalmente, deverão existir duas câmaras virtuais, cada uma responsável por emular a visão de um olho. Portanto, estas câmaras deverão captar a cena com perspetivas ligeiramente diferentes. Para tal, serão usualmente complanares para simplificar o processamento e deverão estar afastadas uma da outra de acordo com a distância ocular do utilizador ou, pelo menos, a distância média para um humano. A esta técnica dá-se o nome de estereoscopia.

Nesta dissertação, o HMD toma um papel importante no controlo do braço robótico, sobretudo porque fornece o referencial do robô ao utilizador, reduzindo qualquer confusão que possa surgir na execução dos comandos gestuais devido à diferença de pontos de vista. Isto é um problema comum na teleoperação de robôs. Um exemplo são instruções do tipo "para cima" quando, na verdade, se pretende instruir o robô para andar "em frente". Devido ao *display* usado, o utilizador poderá ser levado a dar instruções segundo a sua própria perspetiva ("para cima" no ecrã) em vez de considerar a perspetiva do robô ("em frente") (Ellis 2008).

Estes dispositivos podem incluir também um conjunto de sensores, como acelerómetro e giroscópio, para monitorizar os movimentos da cabeça do utilizador. Isto permite deslocar as câmaras que captam a cena virtual ou remota em concordância com estes movimentos, potenciando a sensação de imersão.

Disparidades entre a visão natural do utilizador e a visão simulada, tanto pela última ser demasiado rápida ou conter um atraso, criam desconforto e afetam negativamente esta sensação de imersão. Para além disso, pode induzir enjoo (Bell 2014). O efeito do uso prolongado de equipamentos de realidade virtual já foi estudado e conclui-se que quanto maior a duração da imersão, especialmente após 3 horas, mais severos são os sintomas de enjoo e instabilidade postural (Murata 2004).

Apesar do uso de HMDs em ambientes de realidade virtual originar níveis superiores de telepresença quando comparado ao uso de *displays* 2D comuns, pode também provocar mal-estar que se deve, principalmente, às imperfeições e falta de amadurecimento dos protótipos existentes (Seibert 2014).

2.1.2 Oculus Rift

A versão mais recente deste equipamento de realidade virtual é a Oculus Rift Development Kit 2 (**Figura 2**), dirigida para desenvolvedores[1]. De acordo com a página *web* oficial (Oculus VR 2015), inclui integração com os motores Unreal Development Kit, Unreal Engine 4 e Unity 4, para um desenvolvimento de aplicações e jogos de realidade virtual fácil e rápido. A API presente no SDK suporta, atualmente, desenvolvimento direto em C/C++ (Oculus VR 2014).

[1] Para uma lista de especificações técnicas, ver https://www.oculus.com/dk2/

O Rift possui um *display* OLED de baixa persistência, o que reduz os efeitos de *motion blur* e tremor da imagem. Isto, por sua vez, atenua os sintomas de enjoo e promove a sensação de presença. A resolução por olho é 960x1080, mas possui um único monitor de 1920x2160. Portanto, as imagens renderizadas para os dois olhos devem ocupar, cada uma, uma metade do monitor (Bell 2014).

Figura 2: Oculus Rift DK2.[2]

Uma propriedade das lentes do Oculus Rift é que estas ampliam a imagem de forma a aumentar o campo de visão, induzindo, porém, uma distorção "*pincushion*" (**Figura 3**) (Bell 2014). Para eliminar esta distorção, pode-se aplicar uma distorção "*barrel*" às imagens que passarão pelas lentes e que serão, depois, expostas aos dois olhos. Assim, corrige-se a distorção das lentes e mantêm-se as vantagens da ampliação (Bell 2014).

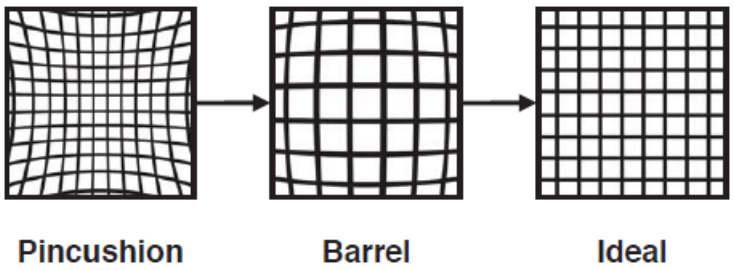

Figura 3: Distorções "*pincushion*" e "*barrel*" (Bell 2014)**.**

O Oculus Rift possui ainda vários sensores. Para medir os movimentos rotacionais da cabeça do utilizador nos três eixos – *yaw*, *pitch* e *roll* – (**Figura 4**), o Oculus Rift SDK usa um giroscópio presente no equipamento para obter a velocidade angular nestes três eixos, um acelerómetro para corrigir erros de deriva de *pitch* e *roll* (LaValle 2013b; Oculus VR 2014), e ainda um magnetómetro para corrigir erros de deriva de *yaw* (LaValle 2013a; Oculus VR 2014). Para monitorizar a posição da cabeça do utilizador num referencial tridimensional (x, y, z), o Oculus Rift serve-se também de uma câmara de infravermelhos (Oculus VR 2015).

[2] Imagem proveniente de https://www.oculus.com/order/

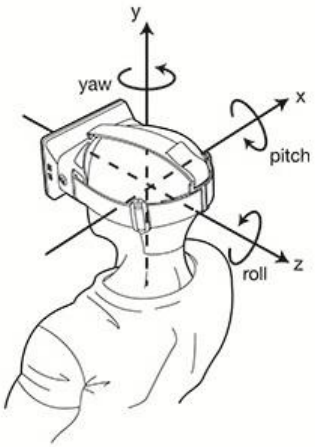

Figura 4: Eixos de rotação da cabeça do utilizador (LaValle 2013b)**.**

2.2 *Body tracking* e sensores de movimento

2.2.1 Introdução

É importante que as interfaces naturais (NUI) sejam fáceis de aprender e exijam interações simples pois, quanto menor for a curva de aprendizagem, melhor (Carvalho 2014). O uso de gestos e poses do corpo como método de interação com sistemas (visualmente) complexos poderá ser preferível a rato e teclado – estes últimos são desenhados para trabalhar num espaço 2D, enquanto os sistemas mencionados geralmente requerem interações num espaço 3D (Bonansea 2009).

Gestos dinâmicos, ou apenas gestos, dependem do movimento realizado. Exemplos disto são *swipe* para esquerda/direita e abrir/fechar a mão. Já os gestos estáticos são poses ou posturas do corpo (Carvalho 2014) como, por exemplo, apontar para a esquerda/direita e mão esticada/fechada.

Existem imensas técnicas e tecnologias para identificar o corpo do utilizador e seguir os seus movimentos. Algumas forçam o utilizador a equipar certos aparelhos: segurar comandos como o Wii Remote Plus da Nintendo e o PlayStation Move da Sony (*handhelds*) e até vestir fatos ou luvas com marcadores que facilitam a deteção das articulações relevantes do corpo (*wearables*). Outras técnicas permitem também ao utilizador executar gestos e poses sem qualquer apetrecho no seu corpo. As tecnologias de *body tracking* e deteção de movimento cuja viabilidade se pretende estudar nesta dissertação são as *wireless*, mãos-livres e de baixo custo.

Uma forma rudimentar de deteção das mãos do utilizador é a segmentação da imagem de uma câmara RGB pela cor da pele, mas esta estratégia sofre severamente de efeitos de oclusão quando os dedos estão curvados ou a mão faz um punho (Carvalho 2014; Chaudhary et al. 2011).

Alguns *handhelds* usam sensores inerciais, como giroscópios e acelerómetros, para avaliar o movimento e rotação dos braços e mãos do utilizador. Outros possuem também marcadores a serem detetados e rastreados através de uma câmara, para obter a posição dos *handhelds* no espaço.

Alternativamente, alguns sistemas calculam a profundidade dos vários objetos do *foreground* da imagem, seguido de métodos de aprendizagem computacional para depois derivar a posição, orientação e movimento dos objetos, nomeadamente do utilizador. Para calcular a profundidade, alguns sistemas usam um projetor de infravermelhos (IV) que projeta um padrão de luzes IV no espaço. Este padrão serve como código luminoso a ser analisado por um sensor de IV. Posteriormente, aplicam-se métodos de estereoscopia para fazer a correspondência entre os dados do projetor e do sensor, e faz-se a triangulação da posição 3D de cada ponto da superfície dos objetos. Este método usa o que é conhecido por sensores de profundidade de luz estruturada (*structured light*) (Hansard et al. 2012). Outro método para calcular a profundidade é o uso de câmaras tempo-de-voo (*time-of-flight*). Esta estratégia baseia-se na medição da diferença de fase entre as ondas IV emitidas e as respetivas ondas refletidas em superfícies, para depois calcular a distância de cada pixel do sensor aos objetos (Hansard et al. 2012). Tanto para os sensores de profundidade de luz estruturada como os de tempo-de-voo, luz IV refletida não é um sinal confiável para todos os tipos de superfícies, como é o caso de materiais especulares e translúcidos (Hansard et al. 2012). Conhecendo a profundidade dos vários pontos da imagem, pode-se então interpolar a localização do utilizador e os seus movimentos ao longo de várias *frames*, através de algoritmos de aprendizagem computacional complexos e usualmente proprietários.

Outros sistemas inovadores usam eletromiografia (EMG) para medir o potencial elétrico gerado pelas células dos músculos, geralmente do antebraço, através de elétrodos à superfície da pele, de forma não intrusiva (Zhang et al. 2011). O nível do sinal da EMG representa diretamente o nível de atividade muscular. Interpretando as leituras obtidas, é possível inferir os movimentos da mão e dos dedos.

2.2.2 Dispositivos comerciais relevantes

2.2.2.1 Kinect for Windows v2

O Kinect for Windows v2 é um sensor de movimento, com tecnologia tempo-de-voo para cálculo de profundidade (Meisner & Knies 2013), uma câmara de cores, um emissor de IV e um *array* de microfones, permitindo saber a localização e movimentos de pessoas, bem como as suas vozes (Microsoft 2015a). Este equipamento tem a capacidade de identificar até 6 pessoas (corpo inteiro) e 25 articulações por esqueleto, com um alcance de 0.5 metros a 4.5 metros. O SDK 2.0 permite desenvolver aplicações para Windows 8.0, Windows 8.1 e Windows Embedded 8, usando linguagens .NET como C++, C# e Visual Basic (Microsoft 2015b).

Figura 5: Kinect for Windows v2.[3]

As articulações reconhecidas pelo Kinect não incluem pontos suficientes das mãos e dedos do utilizador para poder detetar, na íntegra e com precisão, gestos e poses destes membros. Nesta zona, apenas se reconhecem três pontos: um para o centro da mão, um para a ponta do polegar e outro para a ponta acumulada dos outros quatro dedos (Microsoft 2015a).

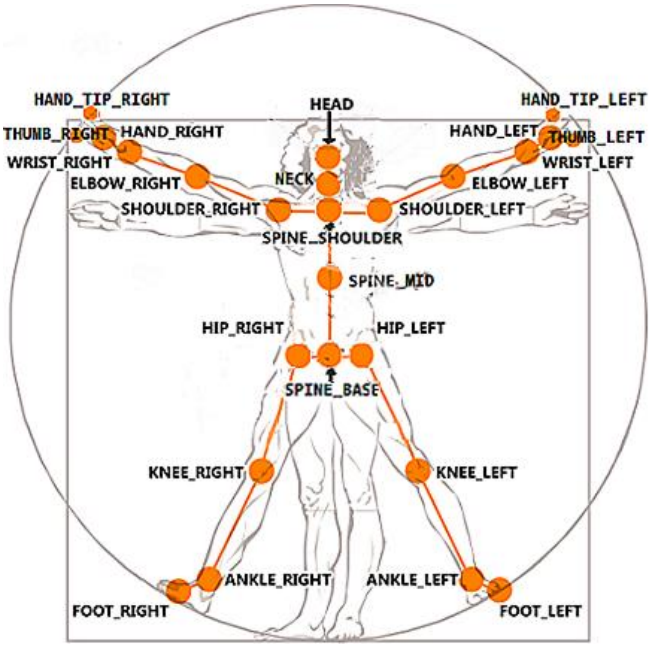

Figura 6: Esqueleto e articulações reconhecidas pelo Kinect for Windows v2.[4]

[3] Imagem proveniente de http://blogs.msdn.com/b/kinectforwindows/archive/2014/03/27/revealing-kinect-for-windows-v2-hardware.aspx
[4] Imagem proveniente de http://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

2.2.2.2 Leap Motion

O Leap Motion é um pequeno sensor de movimento que é capaz de detetar os movimentos das mãos e dedos do utilizador com maior precisão que o Kinect, com uma sensibilidade de até 1/100 de milímetro (Leap Motion 2014b). Este aparelho usa sensores óticos e luz IV, tem um campo de visão de cerca de 150º e um alcance de 2.5 cm a 60 cm (Leap Motion 2014a).

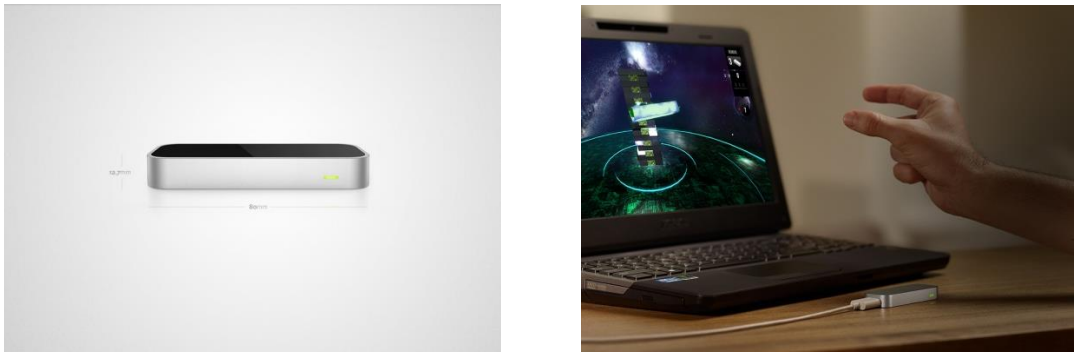

Figura 7: Leap Motion.[5]

As articulações identificadas por este dispositivo são apresentadas na **Figura 8**. O Leap Motion é apropriado para sistemas que pretendem detetar gestos e poses das mãos e dedos do utilizador, enquanto o Kinect destina-se a movimentos do corpo inteiro. É possível ainda executar gestos com uma caneta ou objeto semelhante, para ações mais precisas e movimentos mais intuitivos na realização de tarefas como, por exemplo, desenho.

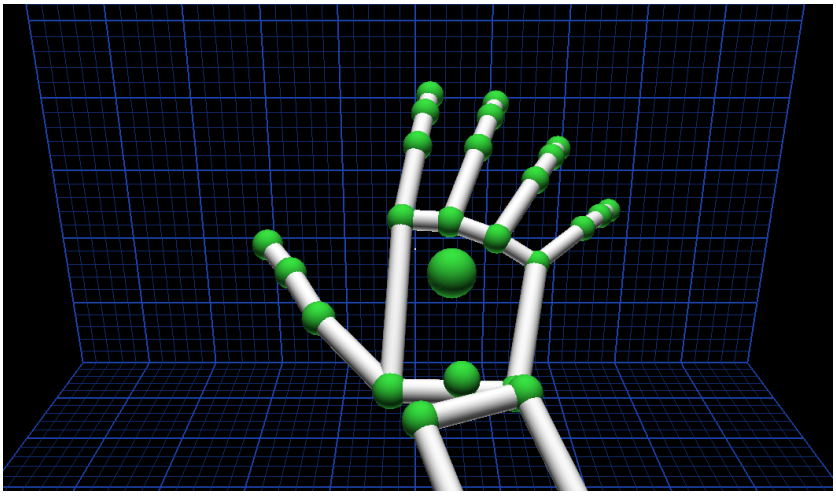

Figura 8: Esqueleto e articulações da mão reconhecidas pelo Leap Motion.[6]

[5] Imagens provenientes de https://www.leapmotion.com/product
[6] Imagem proveniente de https://developer.leapmotion.com/getting-started

É possível desenvolver aplicações que usem o Leap Motion em Objective-C, C++, C#/Unity, Java, JavaScript e Python. As APIs estão bem documentadas e existem vários exemplos de uso para todas as linguagens.

2.2.2.3 Myo

O Myo é um *wearable* que aplica eletromiografia para detetar gestos da mão do utilizador. Este dispositivo é uma leve pulseira que deve ser equipada no antebraço. O Myo contém um giroscópio, um acelerómetro e um magnetómetro para detetar movimentos do antebraço e utiliza sensores EMG para detetar os movimentos e gestos da mão e dedos. É compatível com Windows, Mac, iOS e Android, e comunica com o computador ou telemóvel por Bluetooth (Thalmic Labs 2014).

Figura 9: Pulseira Myo.[7]

Desenvolvedores podem criar aplicações em C++, Objective-C, Java e Lua, existindo ainda *bindings* de linguagens suportadas pela comunidade.

O SDK deste equipamento suporta apenas um pequeno número de poses da mão. Com estas poses e a informação do movimento do braço, é possível implementar o reconhecimento de vários gestos (Goodine 2014). Porém, como não é possível ao programador customizar as poses facilmente e com bons resultados, cria-se uma limitação no número e tipo de gestos detetáveis.

2.2.3 Reconhecimento de gestos personalizados

Esta secção aborda uma área ambiciosa, potencialmente fora do âmbito desta dissertação, mas que foi, mesmo assim, estudada pelo seu valor para trabalho futuro.

[7] Imagem proveniente de https://www.thalmic.com/en/myo/techspecs

Usualmente, os dispositivos existentes reconhecem, à partida, um conjunto reduzido de gestos e poses. Para aplicações complexas e exigentes no que toca à interação com o utilizador, poderá ser necessário definir outros tipos de comandos gestuais.

É importante que o reconhecimento destes comandos seja feito em tempo real, o que implica que o processamento seja o mais simples e rápido possível (Bonansea 2009).

Existem estratégias de classificação que se podem aplicar a este problema. Para tal, é preciso, primeiro de tudo, representar os gestos e poses através de, por exemplo, *feature vectors*. Estes vetores podem conter dados relativos à posição (x, y, z) e orientação (*pitch*, *yaw*, *roll*) de cada mão, bem como a posição e orientação de cada articulação suportada pelo sensor, em cada momento (Yin & Davis 2010; Carvalho 2014). Algumas estratégias requerem um treino prévio do classificador, usando exemplos de treino previamente obtidos (exemplos de execução de gestos).

Um método usado para classificar movimentos como gestos e reconhecer poses é o uso de restrições. Programaticamente falando, define-se o gesto a partir de um conjunto de restrições de movimento, posição ou orientação. Se a leitura obtida pelo sensor estiver em concordância com as restrições, considera-se que esse gesto ou pose foi executado (Carvalho 2014).

Alternativamente, o classificador pode basear-se em *Support Vector Machines* (SVMs). Um SVM tenta separar linearmente os dados de treino (vetores), de acordo com uma classificação binária. Muitas vezes, os pontos não são linearmente separáveis no espaço original. Então, um SVM mapeia os dados num espaço de dimensão superior e encontra o hiperplano que separa linearmente esses dados neste espaço, maximizando ainda a margem entre os pontos de treino mais próximos a esse hiperplano para assegurar uma alta capacidade de generalização do classificador (Vapnik & Cortes 1995). Novos pontos podem ser classificados mapeando-os neste espaço de alta dimensão e verificando em que lado do hiperplano se encontram. Os SVMs são, inerentemente, classificadores binários, pelo que distinguem os dados em apenas duas classes possíveis. No contexto em questão, classes são gestos (ou poses) que se pretendem reconhecer. Para problemas de classificação com mais de 2 classes, como é o caso de um sistema que espera reconhecer mais de dois tipos de gestos, existem duas soluções comuns. Uma solução é criar um SVM para cada par de gestos (*one-versus-one*) e, no momento da classificação de um novo vetor, escolher a classe que é selecionada por mais SVMs (Manning et al. 2008). Outra estratégia é criar um SVM para cada gesto contra todos os outros gestos (*one-versus-all*). Cada um destes SVMs classifica novos pontos como sendo um determinado gesto ou qualquer um dos outros. Nesta estratégia, cria-se então um número de classificadores igual ao número de gestos/poses e, no momento da classificação, escolhe-se a classe que categoriza o ponto com maior margem ao hiperplano do respetivo SVM (Manning et al. 2008). Treinar um SVM implica encontrar a solução de um problema de otimização quadrático muito grande. Existem adaptações ao classificador original, como o *Sequential Minimal Optimization* (SMO), que divide este problema em problemas

quadráticos mais pequenos, sendo mais eficiente tanto em termos de memória como de tempo (Platt 1998).

Outra forma de reconhecer gestos executados pelo utilizador é compará-los geometricamente com instâncias de treino, que servem de modelo (*geometric template matching*). Um algoritmo desta família baseado em "nuvens de pontos", onde os gestos são representados como conjuntos não ordenados de pontos, é o $P[8] (Vatavu et al. 2012).

Correlação cruzada é outro método para comparação de gestos com instâncias de treino, fundamentado na estimação do grau de correlação entre duas séries (Bourke 1996). Estas séries poderão descrever a sequência de posições e orientações que as mãos tomam ao longo de um gesto. O gesto de treino com maior valor de correlação cruzada com o movimento do utilizador será o retorno correto.

Redes neuronais artificiais (RNAs) são outra solução para o problema da aprendizagem e reconhecimento de gestos personalizados. Uma RNA é constituída por imensos elementos de processamento altamente interconectados (Chaudhary et al. 2011). Numa rede simples unidirecional, estes elementos estão organizados em camadas: usualmente, uma camada de *input*, camadas *hidden* e uma camada de *output*. Os elementos da rede podem receber múltiplos dados de entrada de elementos de camadas anteriores e retornar um *output* para vários elementos da camada seguinte (Murakami & Taguchi 1991). As ligações entre elementos têm pesos associados, afetando o peso dos dados de entrada recebidos por um elemento. O treino de uma rede neuronal passa por ajustar o valor destes pesos de forma a minimizar um custo em função do *output* esperado (classificação correta) e do *output* obtido. Neste contexto, a aprendizagem é normalmente supervisionada, usando gestos de treino com informação da classificação correta. Redes neuronais são apropriadas para problemas de reconhecimento de padrões e *data mining* (Chaudhary et al. 2011).

Hidden Markov Models (HMMs) são também frequentemente usados em reconhecimento de padrões, como gestos. Este modelo é composto por uma sequência de estados. Cada estado poderá representar a velocidade, orientação e forma da mão durante uma fase de um gesto (Yin & Davis 2010). Deverá existir um HMM treinado para cada gesto. Uma sequência observada – gesto efetuado pelo utilizador – será então avaliada contra todos os modelos treinados, usando o algoritmo de Viterbi, sendo que a classificação final será dada pelo modelo que classifica essa sequência com maior valor de uma função de probabilidade (Yin & Davis 2010; Chen et al. 2003).

Existem, ainda, métodos alternativos baseados em lógica difusa (*fuzzy logic*) e algoritmos genéticos (Chaudhary et al. 2011), e também estratégias híbridas que incorporam vários métodos anteriormente referidos.

[8] Um demonstrador *online* está acessível em http://depts.washington.edu/aimgroup/proj/dollar/pdollar.html

2.2.4 Semântica dos gestos

Em interfaces naturais baseadas em gestos, a tradução dos movimentos do utilizador para ações no sistema pode ser realizada por imitação ou por meio de gestos simbólicos. No contexto da teleoperação de robôs, tradução por imitação implica comandar o robô para que este se mova de forma idêntica ao utilizador: particularmente, implica que o braço robótico imite os movimentos do braço e mão de quem o opera. Para estas interações, não é necessário o treino e reconhecimento de gestos por parte de um classificador. Por contraste, o uso de gestos simbólicos requer que o sistema consiga classificar os movimentos do utilizador como sendo um determinado gesto previamente treinado. Estes gestos simbólicos não são tomados pelo seu significado literal. Um exemplo é um movimento circular com o dedo indicador para comandar o braço robótico a rodar a articulação do pulso. Neste caso, o significado dado ao gesto não é literal mas deve ser, no entanto, intuitivo. Como alguns movimentos, ordens e ações não são imitáveis pelo utilizador, poderá ser necessário criar estas abstrações gestuais, que devem ser facilmente entendidas e adotadas (Carvalho 2014).

2.3 Análise de tarefas

No desenvolvimento de interfaces de utilizador para sistemas informáticos, desde aplicações *web* a sistemas operativos, é essencial ter em consideração as tarefas que o utilizador pretende realizar e os objetivos que este espera atingir ao interagir com o sistema. Porém, as interfaces são, muitas vezes, altamente inflexíveis. Isto exponencia os problemas de *design* existentes, que se traduzem num fraco suporte à realização eficiente e intuitiva do trabalho do utilizador (Crystal & Ellington 2004). Na área da interação pessoa-computador (*Human-Computer Interaction* ou HCI), procura-se fazer a análise dessas tarefas, o que inclui estudar a atividade humana, perceber como os utilizadores as realizam e descrevê-las em detalhe.

Segundo (Rind et al. 2014), estas tarefas são caraterizadas em 3 dimensões: composição (alto ou baixo nível), abstração (concretas ou abstratas) e perspetiva (problemas ou ações). As tarefas podem ser de alto ou baixo nível, por serem decomponíveis em subtarefas; podem também ser concretas ou abstratas, por serem expressas em termos do domínio do utilizador ou usando termos abstratos, respetivamente; e podem ainda ser problemas ou ações, consoante as tarefas são expressas pelo problema a resolver ou pelas atividades a realizar para resolver esse problema, respetivamente.

O tipo de análise mais comum e de interesse no contexto desta dissertação é *Hierarchical Task Analysis* (HTA). HTA é um método baseado na decomposição hierárquica de tarefas, que as divide em subtarefas e operações unitárias (Crystal & Ellington 2004), e define ainda planos para descrever como, por que ordem e em que circunstâncias executar essas subtarefas e operações. É possível representar esta hierarquia textualmente ou por meio de diagrama. No

contexto deste trabalho e durante o seu desenvolvimento, este tipo de análise é importante para definir as tarefas, subtarefas e ações que o operador do braço robótico pode realizar, bem como os planos que atuam como regras para o controlo adequado do robô.

2.4 Controlo de robôs

Atualmente, a maioria dos robôs industriais é programada e controlada através dos *teach pendants*, um processo moroso que requer conhecimentos técnicos (Neto et al. 2009). É comum interagir por meio de *joysticks* e dezenas de botões. Novas alternativas para controlo de robôs têm surgido recentemente, em especial a implementação de reconhecimento de gestos usando sensores inerciais e abordagens baseadas em visão por computador, possivelmente complementado com reconhecimento da fala. Isto permite a teleoperação de robôs por parte de utilizadores sem experiência técnica. Implementações de sistemas deste género são discutidas na secção **2.5** abaixo.

Existem várias *frameworks* para desenvolvimento de *software* de robótica, que gerem a complexidade e facilitam a rápida prototipagem do sistema (Quigley et al. 2009). Um deles é o *Robot Operating System* (ROS), uma *framework open-source* multilinguagem que fornece uma coleção de ferramentas, bibliotecas e convenções, suportando um vasto conjunto de plataformas robóticas (Open Source Robotics Foundation 2015). O ROS proporciona uma camada de comunicação *peer-to-peer* entre vários processos, possivelmente em *hosts* diferentes, o que permite delegar computação intensiva a máquinas externas. Esta *framework* reutiliza código de outros projetos *open-source*, como *drivers*, sistemas de navegação, simuladores como o Gazebo[9], algoritmos de visão por computador e de planeamento de trajetórias (Quigley et al. 2009). A nomenclatura adotada resume-se a nós, mensagens, tópicos e serviços. Nós são processos que computam algo e comunicam entre si através de mensagens (estruturas de dados), seguindo o padrão *publish-subscribe*. Um nó envia uma mensagem publicando num tópico identificado por uma simples *string*; um nó interessado nesse tópico pode subscrevê-lo (podem existir múltiplos publicadores e subscritores concorrentes para um mesmo tópico) (Quigley et al. 2009). Esta estratégia de *broadcasting* pode não ser adequada para transações síncronas, pelo que um nó pode então fornecer um serviço, definido por uma *string* identificativa e um par de tipos de mensagens – pedido e resposta –, funcionando de forma análoga a um serviço *web* ou a *Remote Procedure Calls*, em que apenas um nó é responsável por um dado serviço com um determinado nome (Quigley et al. 2009). Para suportar desenvolvimento colaborativo, o *software* do ROS é organizado em pacotes. Existem pacotes para uma multitude de funcionalidades, de entre as quais: identificação de objetos, navegação, planeamento de percursos, mapeamento e reconhecimento de faces e gestos. Um exemplo é o pacote RViz: uma

[9] http://wiki.ros.org/gazebo

ferramenta de visualização 3D que permite seguir o estado de um robô e visualizar trajetórias planeadas.

Outro *software* de interesse é o MoveIt!, criado sobre o ROS base, que integra funcionalidades avançadas de planeamento de percursos, manipulação, perceção 3D, cinemática, verificação de colisões, controlo e navegação (Sucan & Chitta 2015). O MoveIt! fornece interfaces de alto e baixo nível para expor estas funcionalidades, servindo não só como uma abstração mas também como uma expansão do ROS base.

2.5 Trabalhos relacionados

Existem vários trabalhos produzidos na área de controlo de robôs através de gestos e poses.

Em (Neto et al. 2009), Neto et al. descrevem um sistema que utiliza dois acelerómetros de 3 eixos *wireless* montados nos braços do utilizador para capturar os seus movimentos – gestos e poses dos braços –, para controlar um robô industrial. É usada uma RNA treinada com o algoritmo *back-propagation*, com a qual obtiveram uma taxa de reconhecimento de 92%. Procuraram ainda obter tempos de resposta baixos (160 ms). Este trabalho limitou-se a detetar movimentos dos braços, a partir dos acelerómetros, não considerando movimentos das mãos e dedos. Isto não só restringe o número e tipo de gestos identificáveis, como dificulta a interação com o sistema se se pretender reconhecer um número elevado de comandos diferentes – torna-se difícil distinguir os vários gestos se estes forem muitos. Ao mesmo tempo, estas restrições tornam a interação menos natural pois gestos e poses de apenas os braços do utilizador nem sempre conseguem reproduzir o significado de certas ações do robô, como agarrar e largar objetos. Adicionalmente, este trabalho não considerou a visualização do espaço onde o robô se move e realiza os comandos dados, no caso do utilizador se encontrar noutro local ou se não tiver visão e perspetiva adequadas desse espaço.

Em (Waldherr et al. 2000), Waldherr et al. descrevem uma interface gestual baseada em visão por computador para controlo de um robô móvel, equipado com uma garra. Neste trabalho, é usada uma câmara para seguir uma pessoa e reconhecer gestos e poses dos braços. O robô segue essa pessoa num ambiente de escritório, com intensidades de luz variáveis, e recebe comandos para tarefas de limpeza, interpretando movimentos dos braços. A pessoa guia, então, o robô a locais específicos e ordena-o a apanhar lixo, apontando para objetos no chão. Para a deteção rápida da localização da pessoa, foi usada uma estratégia de segmentação das cores da sua cara e da roupa vestida na parte superior do corpo. Posteriormente, foram usados dois métodos alternativos para reconhecimento de gestos: um baseado em comparação de modelos (*template matching*) e outro em redes neuronais. Os resultados foram muito semelhantes para ambos os métodos, não havendo um que se destacasse. A taxa de erro global foi de 3%. Porém, Waldherr et al. julgam que a combinação destes dois métodos de reconhecimento de gestos daria ainda melhores resultados. Esta implementação assumia que a cara da pessoa era visível

em todos os momentos e que a pessoa mantinha uma distância ao robô fixa. Para além disso, não era expectável que esta interface fosse suficientemente robusta para funcionar em ambientes lotados de pessoas. A limitação de maior interesse para esta dissertação é o facto de apenas detetar os gestos e poses dos braços do utilizador e não detetar movimentos mais delicados dos dedos. Esta limitação advém principalmente da estratégia de captura – imagens de cor de uma pessoa a 2-3 metros de distância – e também dos métodos de visão por computador usados para detetar a pessoa e as poses realizadas, que tinham de ser pouco exigentes dado o poder computacional limitado do robô móvel.

Existem outros trabalhos que capturam os movimentos com maior detalhe, fazendo o reconhecimento de gestos e poses não só dos braços como também das mãos e dedos, realizados em contextos bastante diferentes desta dissertação como, por exemplo, a interpretação de linguagem gestual. Em (Zhang et al. 2011), Zhang et al. descrevem uma *framework* para reconhecimento de gestos usando acelerómetros e sensores EMG. São usados HMMs e uma árvore de decisão no motor de reconhecimento. Com isto, conseguiram classificar um subconjunto de 72 palavras da linguagem gestual chinesa, com uma taxa de reconhecimento superior a 90%. Em (Murakami & Taguchi 1991), Murakami e Taguchi descrevem um método para classificar um subconjunto de 42 símbolos da linguagem gestual japonesa, através de uma RNA recorrente (cíclica), possibilitando o reconhecimento de gestos contínuos.

Nesta dissertação, procura-se aplicar algumas destas estratégias já experimentadas e comprovadas no passado para fazer agora a deteção de gestos e poses subtis das mãos e dedos do utilizador, de forma a controlar um braço robótico com precisão. Adicionalmente, considera-se o problema da visualização do espaço onde o robô opera, resolvendo-o por meio de HMDs.

2.6 Conclusões da revisão

As interfaces naturais que se perseguem nesta dissertação têm vindo a ganhar ímpeto nos últimos anos. Desde HMDs a sensores de movimento e reconhecimento de gestos, tem ocorrido uma evolução rápida destas tecnologias que possibilitam uma interação transparente e intuitiva.

De entre os dispositivos existentes, dada a facilidade de acesso, custo, maturidade da tecnologia e suporte comunitário, opta-se por usar o Oculus Rift e o Leap Motion na solução proposta. Devido à necessidade de detetar movimentos subtis das mãos e dedos do utilizador, o Kinect não é de todo adequado neste contexto. A pulseira Myo não suporta uma fácil customização das poses detetáveis por parte do programador, o que limita o número de gestos estáticos e dinâmicos que o utilizador pode usar para interagir com o sistema.

É essencial aplicar algoritmos simples mas eficazes no reconhecimento de gestos em tempo real. A implementação de gestos personalizados, para complementar os já suportados pelo sensor de movimento usado, é crucial em sistemas que requerem um grande número de ordens ou interações do utilizador, cada uma representada por um gesto diferente. Neste trabalho

experimental, isto não se verifica forçosamente mas reconhece-se o seu potencial na teleoperação de robôs.

A escolha da *framework open-source* ROS para o desenvolvimento de *software* robótico deve-se à modularidade e contribuição comunitária na forma de pacotes, com as mais variadas funcionalidades, e ainda à integração com bibliotecas externas.

Capítulo 3

Sistema de Realidade Virtual e de Interação Natural por Gestos para Controlo de Braço Robótico

Neste capítulo, faz-se uma descrição detalhada do problema em mãos e da abordagem proposta para o resolver.

3.1 Descrição do problema

As interfaces e mecanismos comuns para programação e controlo de robôs requerem conhecimentos técnicos, como é o caso dos *teach pendants* para robôs industriais, ou não se adequam diretamente ao mundo tridimensional, como os *joysticks*. Procura-se, então, um método alternativo para controlar um braço robótico de forma fácil e natural e com precisão suficiente para interagir com objetos de pequena dimensão. Isto pode incluir ações como agarrar e pousar objetos, encaixar e desencaixar peças e até mesmo premir botões. Existem vários cenários onde uma alternativa deste tipo poderá ser vantajosa, tanto pela inerente rápida aprendizagem de utilização como pela conveniência que um robô facilmente controlado traz na execução remota de tarefas. Um exemplo é a realização de certas tarefas de manutenção de equipamentos de rede, potencialmente à distância, como ligar e desligar cabos de rede. Este cenário é usado neste trabalho para ajudar a definir uma visão da solução ideal, mesmo que inatingível. Neste exemplo em específico, as interações com objetos a considerar podem reduzir-se a agarrar, largar, encaixar e desencaixar cabos de rede RJ-45 ou de alimentação e premir botões para, então, reconfigurar a topologia física de uma rede ou ligar/desligar dispositivos.

Pretende-se medir a viabilidade das tecnologias referidas nos capítulos anteriores no contexto da teleoperação de robôs, pelo que se procura criar um método para realizar tarefas deste género controlando um braço robótico através de comandos gestuais. Este problema global divide-se em três componentes.

Primeiramente, tenciona-se resolver o subproblema da visualização do espaço, possivelmente remoto, onde se encontram os objetos com os quais interagir ou os equipamentos a operar, de forma a facilitar a execução das tarefas. O uso de monitores 2D comuns é desvantajoso devido à diferença de perspetivas entre o utilizador – que observa um ecrã – e a câmara – que filma o robô e os objetos. Isto torna a execução dos comandos gestuais mais confusa porque o utilizador é obrigado a abstrair-se do seu ponto de vista. É, por isso, importante arranjar uma forma de partilhar a perspetiva, ou referencial, do robô com o utilizador.

O segundo subproblema é a deteção dos movimentos, leitura da posição e reconhecimento de vários gestos e poses do utilizador, nomeadamente dos seus braços, mãos e dedos.

Existe, também, o subproblema do controlo adequado dos movimentos do braço robótico para executar tarefas semelhantes às mencionadas acima com a precisão necessária. As ações do robô resumem-se a alterações da posição e orientação da garra robótica e a movimentos de abertura e fecho desta garra. Surge, ainda, o desafio de obter e manter com fiabilidade a posição desejada da garra robótica para depois interagir com o ambiente. Existe, portanto, a problemática da precisão: os movimentos de translação da garra devem estabilizar ou abrandar a mando do utilizador, por exemplo, quando se aproxima de um objeto, como um cabo de rede no caso do cenário exemplificativo referido. É de notar que o movimento da base do robô de modo a aproximar-se dos objetos é um problema de navegação de bases móveis, que não é abordado nesta dissertação.

3.2 Solução proposta

Pretende-se desenvolver um sistema imersivo com uma interface natural para controlar um braço robótico, idealmente de forma remota, através de comandos gestuais dos braços, mãos e dedos do utilizador. A intenção é realizar tarefas simples com objetos de pequena dimensão – não mais do que 10 centímetros de comprimento –, como fazer a manutenção de equipamentos de rede, organizar e arrumar objetos ou quiçá trabalhar em espaços inóspitos a um ser humano, potencialmente à distância.

No caso de um espaço remoto, para resolver o problema da visualização dos objetos com os quais interagir usa-se um Oculus Rift e uma câmara estereoscópica (3D), ou duas câmaras 2D banais lado a lado, de modo a captar e ver a perspetiva do robô. Com a câmara montada no topo do robô, faz-se *stream* de vídeo estereoscópico em tempo real para o Rift, que deverá ser

equipado pelo utilizador. Isto é responsabilidade de uma aplicação que será referida neste documento como *Oculus App*. O ambiente imersivo criado permitirá a este ver o espaço pelo ponto de vista do robô – como se o braço robótico fosse o seu –, o que deverá promover um controlo por gestos menos confuso e mitigar a sensação de desorientação que o utilizador poderia de outro modo sentir.

Para fazer a monitorização da posição das mãos e o reconhecimento de gestos e poses realizados pelo operador do braço robótico, usa-se o sensor de movimento Leap Motion, pousado numa mesa em frente ao utilizador. A posição da sua mão direita é continuamente reportada ao robô várias vezes por segundo (FPS do sensor é variável) por uma aplicação que se encarrega de subscrever à API do Leap Motion para ser notificada de eventos e receber *frames* com essa informação – esta aplicação será referida de agora em diante como *Leap App*. A posição da mão direita, que controla o braço robótico por imitação, é enviada por *socket* UDP a uma outra aplicação responsável por transformar as coordenadas que estão no referencial do sensor para o referencial do robô e ainda por planear e executar a trajetória que leva a garra da sua posição atual para a posição desejada, através da *framework* ROS e do *software* MoveIt! – esta aplicação será referida neste documento como *Robot App*. Para tal, define-se um paralelepípedo reto completamente contido na região de deteção do sensor, que tem a forma de uma pirâmide invertida com centro no centro do dispositivo, e define-se também outro paralelepípedo reto que delimita a região alcançável pela garra robótica. A transformação das coordenadas resume-se depois a uma transformação geométrica do primeiro prisma para o segundo, com atenção ao facto de o Leap Motion trabalhar em milímetros e o ROS em metros. A solução inclui ainda um modo de precisão ou *fine-tuning* para controlar meticulosamente a posição da garra. Este modo mantém-se ativo enquanto a mão esquerda do utilizador for detetada com o punho fechado, um gesto secundário adicional. Esta funcionalidade da *Leap App* altera as posições enviadas à *Robot App*, reduzindo para metade o efeito das translações da mão direita do utilizador, o que permite realizar movimentos do robô mais precisos ou estabilizar a garra por completo sem que o utilizador tenha de reproduzir esse nível de precisão com os seus próprios movimentos. Quando este modo é desativado, a posição da garra é lentamente ressincronizada com a posição que deveria ocupar caso este modo não tivesse sido ativado, de modo a reajustar a sua posição de acordo com a da mão humana. Juntamente com a posição, a *Leap App* envia também a orientação da mão e braço direitos do utilizador a cada momento, na forma de *roll*, *pitch* e *yaw*. Estes ângulos são usados pela *Robot App* para calcular a orientação equivalente da garra robótica. Para além disso, envia-se repetidamente a distância entre a ponta do polegar e os restantes dedos da mão direita do utilizador para que a segunda aplicação possa abrir a garra em concordância.

Idealmente, o conjunto de tarefas que se pode realizar com o braço robótico é apenas restringido pelas ações básicas do robô, que são mover, rodar, inclinar, abrir e fechar a garra.

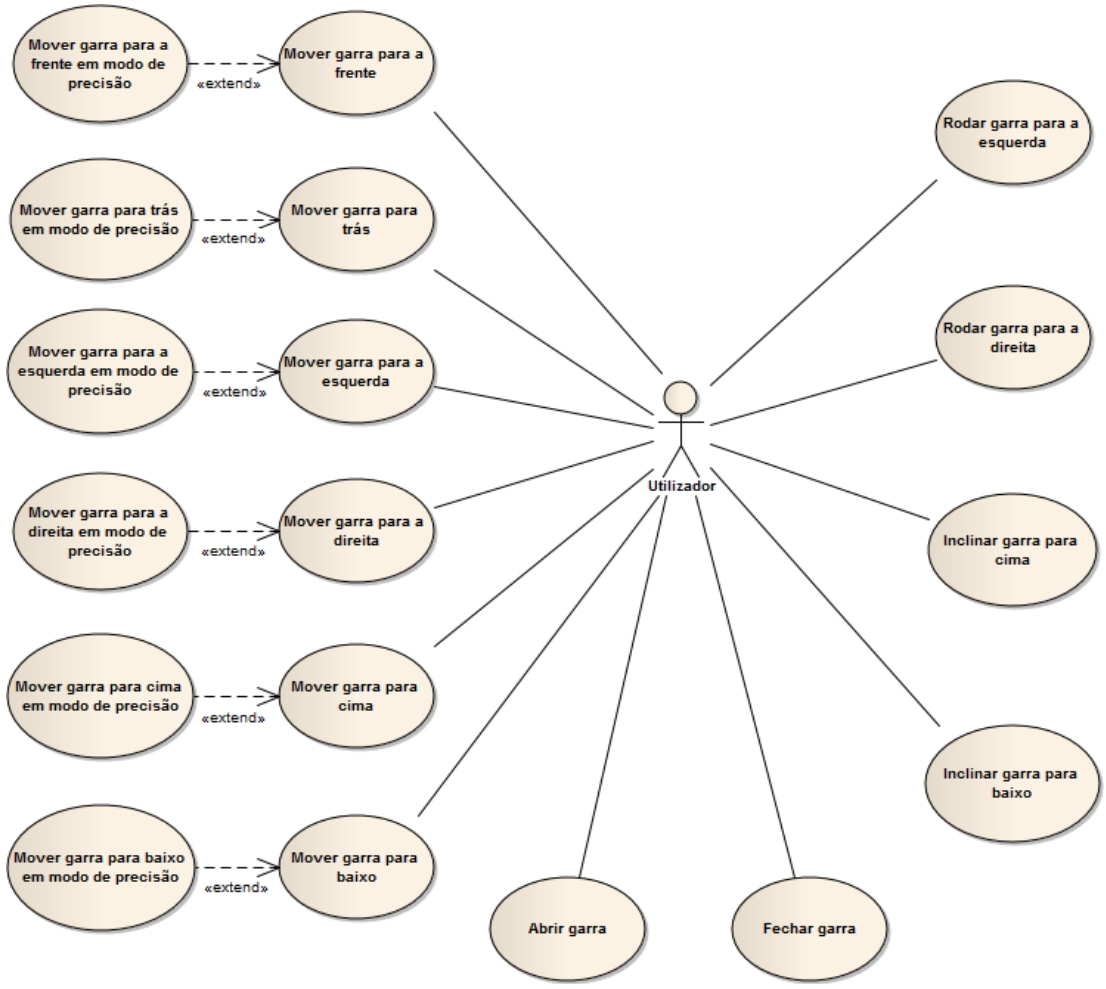

Figura 10: Casos de utilização para o controlo do robô.

Resumindo, o panorama é o seguinte: o utilizador está equipado com o Oculus Rift a ver uma *stream* de vídeo 3D em tempo real com a perspetiva do robô, e o Leap Motion está à sua frente numa mesa; o sistema do lado do utilizador monitoriza a posição, orientação e gestos das mãos do utilizador e comunica essa informação – ordens – à aplicação remota responsável pelo controlo do braço robótico; por sua vez, esta segunda aplicação faz o planeamento necessário e comunica com o robô através da camada de comunicação do ROS para executar as trajetórias planeadas.

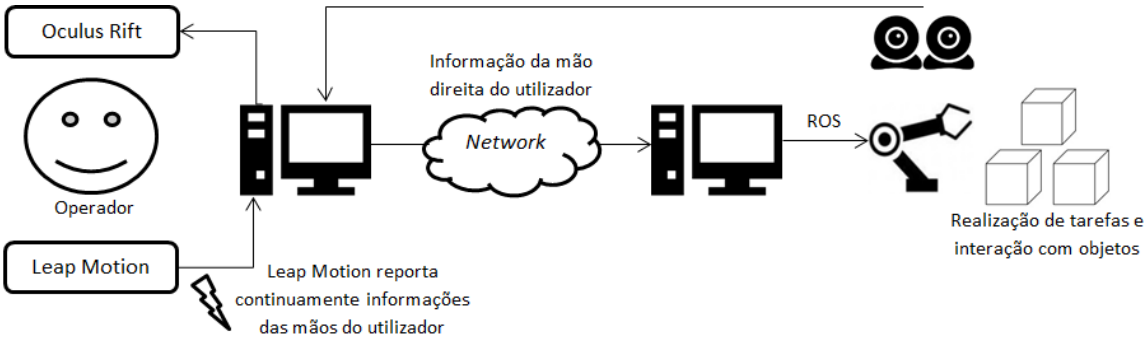

Figura 11: Arquitetura física do sistema.

Capítulo 4

Implementação e Resultados

Neste capítulo, descreve-se a implementação das aplicações que constituem a solução proposta, referem-se os testes e inquéritos efetuados e discutem-se os seus resultados.

4.1 Detalhes de implementação

A *Oculus App*, implementada em C++, utiliza a biblioteca de visão por computador OpenCV[10] (v2.4.9) pelo seu suporte a captura de vídeo com a classe *VideoCapture* e ainda as seguintes bibliotecas e *wrappers* de OpenGL: OGLplus[11], GLEW[12] e GLFW[13]. Evidentemente, esta aplicação também se serve do Oculus Rift SDK para Windows[14] (v0.4.4). O código é baseado nos exemplos *online* do livro *Oculus Rift In Action*[15].

As duas câmaras USB utilizadas têm uma resolução de 640x480 cada, mas a aplicação está preparada para qualquer resolução. As câmaras 2D idênticas – olhos esquerdo e direito – são dispostas lado a lado e no mesmo plano vertical. A distância de uma lente à outra deve estar entre 6 a 6.5 centímetros, que é uma aproximação da distância ocular média de um ser humano.

[10] http://opencv.org/
[11] http://oglplus.org/
[12] http://glew.sourceforge.net/
[13] http://www.glfw.org/
[14] https://developer.oculus.com/
[15] https://github.com/OculusRiftInAction/OculusRiftInAction

Figura 12: Câmaras dispostas lado a lado.

Depois de todas as bibliotecas estarem inicializadas, incluindo o carregamento de *shaders*, a *Oculus App* abre as duas câmaras para leitura. Depois, inicia-se duas *threads* em segundo plano para, continuamente, ler e guardar *frames* destes dois dispositivos, com uma frequência que varia consoante as especificações das câmaras. Ao mesmo tempo e para cada *frame*, é detetada a pose da cabeça através do Oculus Rift, para mais tarde renderizar as imagens orientadas de tal maneira que o utilizador veja o vídeo 3D com uma inclinação nula em relação aos seus olhos. É de notar que, no cenário idealizado, ao contrário de câmaras virtuais em mundos gerados computacionalmente, a orientação das câmaras não se altera segundo os movimentos do Oculus Rift porque estas estão assentes no robô sem nenhum mecanismo físico que as reoriente. Enquanto as *frames* das câmaras são lidas e guardadas em estruturas do OpenCV (matrizes), a *thread* principal usa-as para, continuamente, formar texturas OpenGL a partir dessas imagens e renderizá-las no Oculus Rift com a distorção *barrel* necessária, passando essas texturas à API do HMD.

A *Leap App*, implementada em C#, utiliza uma *framework* multimodal de deteção de gestos criada para os projetos InMERSE dos quais este trabalho faz parte. Esta *framework* encarrega-se de abstrair e generalizar vários sensores de movimento, como Kinect, Leap Motion e Myo, o que permite programar aplicações de interação natural independentemente da API do dispositivo subjacente. De qualquer das formas, a *framework* continua a dar acesso às estruturas de dados particulares do sensor utilizado, se assim o desenvolvedor o desejar. Existem dois modos de funcionamento desta *framework*: aquisição de *frames* – que contêm, entre outras informações, a posição e orientação de certas partes do corpo do utilizador em cada momento – e deteção de gestos – que abstraem a informação de sequências de *frames* para representar um movimento de alto nível.

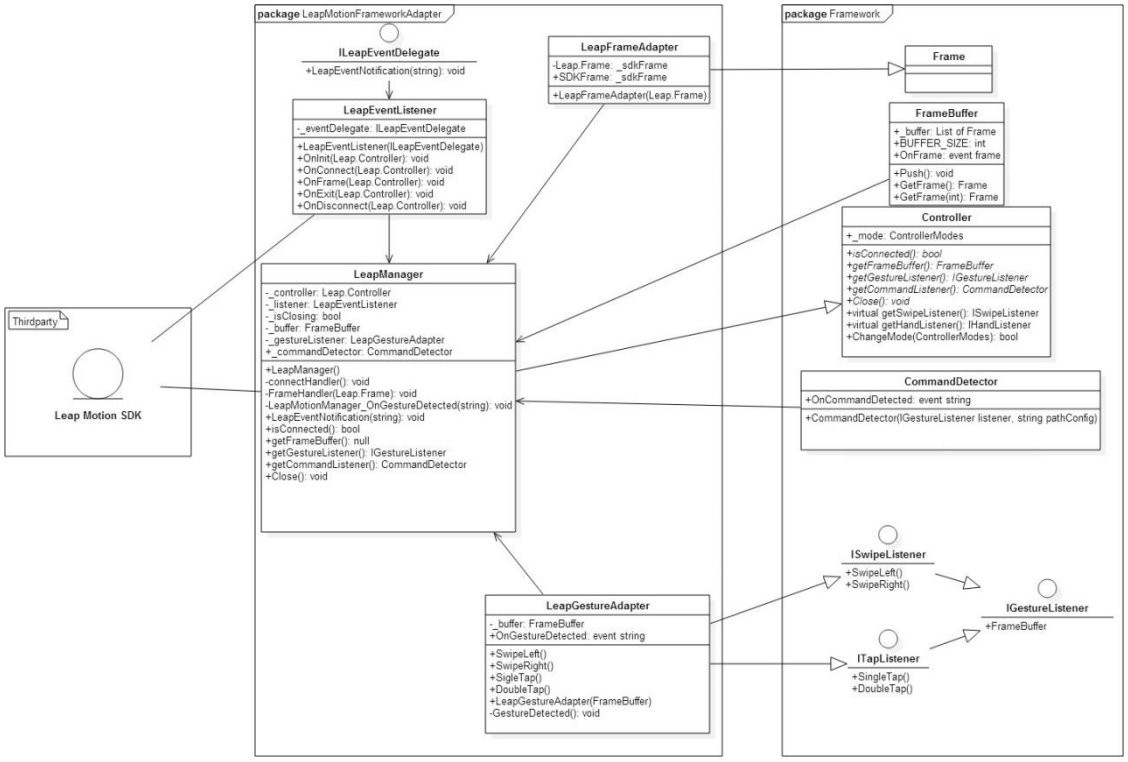

Figura 13: *Framework* multimodal para deteção de gestos (componente do Leap Motion).

Depois de inicializada esta *framework* em modo de aquisição de *frames* e com a sua componente do Leap Motion, a *Leap App* pede o endereço IP e a porta por onde a *Robot App*, já em execução, pode ser contactada. Inseridos esses dados, a *Leap App* subscreve ao evento *OnFrame* da *framework*, para receber novas *frames* assim que a API do sensor as constrói, com uma frequência que varia consoante os recursos computacionais disponíveis mas normalmente contida entre as 20 e as 200 *frames* por segundo. No *handler* desse evento, a aplicação calcula e guarda num objeto os dados a enviar por *socket* UDP à *Robot App*. Esse objeto mantém apenas os dados relativos à *frame* mais recente. Paralelamente, uma *thread* em segundo plano envia, com um intervalo de um segundo entre cada pacote, os dados mais recentes no momento do envio, em formato JSON através desse *socket*. Isto significa que a *Robot App* não recebe informação com a mesma frequência com que o Leap Motion gera *frames*, pelo que algumas destas são efetivamente ignoradas de modo a evitar que a *Robot App* seja obstruída com demasiados pedidos, já que o planeamento e execução das trajetórias do braço robótico são o *bottleneck* do subsistema *Leap App*-*Robot App*.

O cálculo dos dados a enviar, feito no *handler* do evento *OnFrame*, passa por: obter a orientação (*roll*, *pitch* e *yaw*) do pulso direito do utilizador, que será depois usada no cálculo da orientação da garra; calcular a abertura da mão direita, dada pela distância entre o polegar e os restantes dedos da mesma mão, que será usada para determinar a abertura da garra; e obter a posição da palma da mão direita, para depois a *Robot App* poder calcular a posição equivalente da garra. Para além disso, se a mão esquerda for detetada na *frame* com o punho fechado, a

Leap App entra em modo de precisão, reduzindo o efeito das translações da mão direita para metade, enquanto esse modo estiver ativo. Este gesto estático foi programado através da definição de restrições da posição e orientação dos dedos da mão esquerda. Neste modo, no *handler* do evento *OnFrame*, calcula-se a translação entre a *frame* atual e a anterior para então calcular a posição a enviar, que resulta de somar metade dessa translação à posição da *frame* anterior. Depois, quando em modo normal, há um processo de ressincronização que, *frame* a *frame*, corrige lentamente as várias posições a enviar de modo a anular suavemente o desfasamento criado pelo modo de precisão.

A *Robot App*, implementada em C++ para sistemas Unix, requer a instalação do ROS base e do MoveIt!, para usufruir da interface de planeamento *MoveGroup*. A implementação focou-se no robô PR2, ilustrado na **Figura 14**, dada a completude da documentação e a abundância de exemplos do ROS e do MoveIt! orientados a este robô, o que permitiu acelerar a implementação desta parte do sistema e facilitou a obtenção de resultados relevantes em tempo útil. Para além disso, este robô mostrou-se suficiente para atingir o objetivo de validação e experimentação das tecnologias aqui estudadas e das interfaces naturais em cenários de teleoperação de robôs. Este robô tem dois braços com garras de dois dedos, mas apenas o braço direito é controlado pelo utilizador na prova de conceito experimental criada[16].

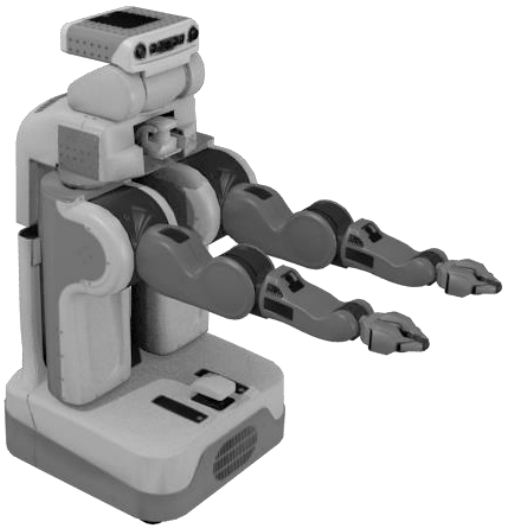

Figura 14: PR2.[17]

Esta aplicação começa por inicializar o nó ROS implementado, que é responsável por traduzir e retransmitir os comandos recebidos da *Leap App* aos serviços ROS de planeamento e execução de trajetórias do robô. Isto inclui, entre outras tarefas, inicializar o RViz – uma ferramenta de visualização 3D para ROS –, e criar o *socket* por onde serão recebidos os dados relativos à posição, orientação e abertura da mão direita do utilizador. Antes de esperar por

[16] https://www.willowgarage.com/pages/pr2/specs
[17] Imagem proveniente de http://www.openrobots.org/morse/doc/1.2/_images/pr2.png

pacotes, a *Robot App* comanda o braço esquerdo do robô a mover-se para a esquerda, de maneira a aumentar o espaço de manobra do outro braço, que será controlado por gestos. Quando um pacote é recebido, a informação nele contida é processada para obter os valores equivalentes admitidos pelo ROS ou pelo MoveIt!. As coordenadas da posição da palma direita do utilizador, previamente normalizadas entre 0 e 1, são convertidas para as gamas de valores alcançáveis pela garra robótica direita. Estas gamas foram obtidas empiricamente e são listadas na **Tabela 1**.

Tabela 1: Gamas de valores usadas para o cálculo da posição da garra robótica direita.

Eixo (referencial do robô)	*Limite inferior (m)*	*Limite superior (m)*
x	0.2	0.67
y	-0.8	0.3
z	0.45	1.2

É também necessário ter em conta as diferentes direções dos eixos nos referenciais do robô e do Leap Motion, como é descrito na **Tabela 2**.

Tabela 2: Equivalência entre as direções dos eixos do Leap Motion e do robô.

Origem		*Eixos*		*Direção*
Leap Motion	*Robô*	*Leap Motion*	*Robô*	
Centro da superfície do dispositivo	x = 0 nos ombros, y = 0 entre os braços, z = 0 na face inferior da base	-z	x	Frente
		-x	y	Esquerda
		y	z	Cima

O processamento da informação de um pacote recebido passa depois por converter a orientação do pulso direito do utilizador para a orientação da garra, em termos dos ângulos *roll*, *pitch* e *yaw* e de acordo com as equações seguintes:

$$\boldsymbol{roll_{garra}} = \begin{cases} \frac{3\pi}{2} - \boldsymbol{roll_{pulso}}, & \boldsymbol{roll_{pulso}} \geq \frac{\pi}{2} \wedge \boldsymbol{roll_{pulso}} \leq \boldsymbol{\pi} \\ -\frac{\pi}{2} - \boldsymbol{roll_{pulso}}, & \boldsymbol{nos\ outros\ casos} \end{cases} \quad \textbf{(1)}$$

$$\boldsymbol{pitch_{garra}} = -\boldsymbol{pitch_{pulso}} \quad \textbf{(2)}$$

$$\boldsymbol{yaw_{garra}} = \boldsymbol{0} \quad \textbf{(3)}$$

Tanto para o Leap Motion como para o robô, estes três ângulos estão no intervalo [-π, π]. A conversão representada por estas equações é necessária dado que as APIs usadas definem estes ângulos com origens e sentidos diferentes, para um mesmo vetor. O *yaw* da garra robótica é

definido pela *Robot App* como sendo constante e nulo, para tornar o controlo mais simples e mais resistente a rotações da mão do utilizador pouco significativas ou mesmo involuntárias. Para além disso, um *yaw* nulo ajuda a manter a pose de todo o braço robótico o mais semelhante possível à respetiva pose do utilizador: a articulação do pulso do PR2 não roda no plano horizontal, pelo que um *yaw* diferente de zero, para uma dada posição da garra, só é conseguido com alteração das outras articulações do braço robótico, divergindo então da pose do braço do utilizador. Note-se que a opção de manter este ângulo nulo não tem qualquer efeito na posição da garra em relação à posição da palma direita do utilizador.

O processamento da informação de um pacote recebido pela *Robot App* termina com a conversão da abertura da mão direita do utilizador, previamente normalizada entre 0 e 1, para a abertura da garra. A gama de valores válidos para a articulação da garra do PR2, obtida empiricamente, é [0, 0.55].

Feito o processamento dos dados do pacote, a *Robot App* define a posição e orientação anteriormente convertidas como sendo o objetivo a atingir para a garra robótica e usa a interface *MoveGroup* disponibilizada pela *framework* MoveIt! para planear e executar a trajetória que leva a garra da sua posição e orientação atuais para as desejadas. Similarmente, a aplicação define a abertura calculada como sendo o objetivo a atingir para a articulação encarregue de abrir e fechar garra e planeia e executa a trajetória para essa articulação.
As trajetórias planeadas e as posições e orientações finais, após cada pedido da *Leap App*, podem ser visualizadas na ferramenta RViz com uma representação 3D do PR2.

4.2 Testes efetuados e discussão dos resultados

4.2.1 Oculus App

Um exemplo de execução da *Oculus App* pode ser visto na **Figura 16**, a respeito da cena fotografada na **Figura 15**.

Figura 15: Cena-exemplo filmada pelas câmaras.

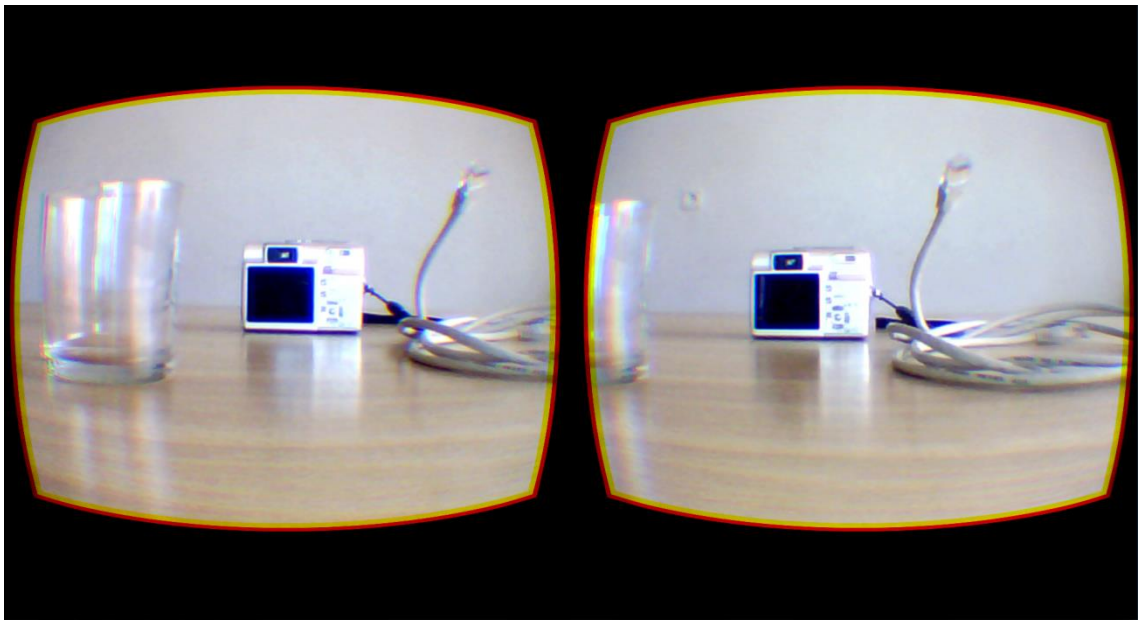

Figura 16: Exemplo de resultado do processamento da *Oculus App* e *input* para o Rift.

Esta aplicação foi testada por 46 utilizadores numa sala com mesas e computadores. Todos exceto 2 conseguiram usar o Oculus Rift; estes 2 removeram de imediato o aparelho por lhes causar tonturas. Dos 44 restantes, 2 viram duas imagens sobrepostas (desfasamento) em vez de uma imagem tridimensional. Os restantes 42 notaram a profundidade da cena a ser filmada e conseguiram perceber a tridimensionalidade dos objetos para distâncias às câmaras entre os 10 centímetros e os 2 metros como, por exemplo, mãos e canetas dispostas à frente das câmaras. Este intervalo de distâncias cobre o espaço de interação de um robô como o PR2, para uma dada posição da sua base. Objetos mais distantes, como computadores no fundo da sala, eram vistos corretamente por estes 42 utilizadores mas a sua profundidade não era notada. A fraca qualidade de imagem e o baixo *frame rate* foram outras observações feitas. Isto dificulta a perceção do espaço, mas tal deve-se apenas às câmaras usadas na prova de conceito, que podem ser facilmente substituídas. Para além disso, o atraso causado pelo processamento da *Oculus App* não é percetível, por exemplo entre movimentos das câmaras ou mudanças na cena e a respetiva visualização através do Oculus Rift.

Com os testes efetuados, notou-se ainda a tolerância do cérebro humano à disparidade entre a distância interpupilar do utilizador e a distância entre as duas câmaras. A configuração destas não tem de corresponder exatamente às caraterísticas físicas do utilizador.

Os HMDs permitem discernir melhor a posição dos objetos relativamente a outros objetos e às câmaras, em comparação com monitores 2D. Isto deverá, então, facilitar o controlo do braço robótico, nomeadamente o reposicionamento da garra nas proximidades de objetos, visto que o utilizador consegue avaliar melhor a distância entre estes e a garra, até haver colisão.

4.2.2 *Leap App* e *Robot App*

Nas figuras que se seguem, apresentam-se exemplos de execução do sistema de controlo do braço robótico. De um lado, tem-se a pose do braço e mão direitos do utilizador relativamente ao Leap Motion e, do outro, a visualização do robô na ferramenta RViz. É possível constatar diferentes posições e orientações da mão e as respetivas poses do robô: garra posicionada para a esquerda, direita, frente, cima ou baixo; garra a apontar para cima ou para baixo (*pitch*); garra na vertical ou na horizontal (*roll*); e garra aberta ou fechada.

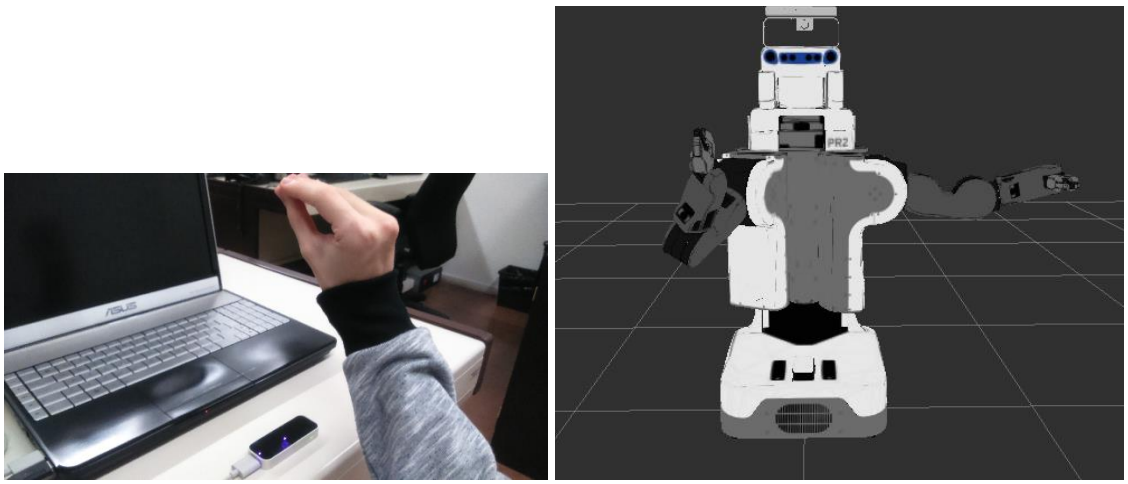

Figura 17: Controlo do robô - garra para a frente, a apontar para cima e fechada na vertical.

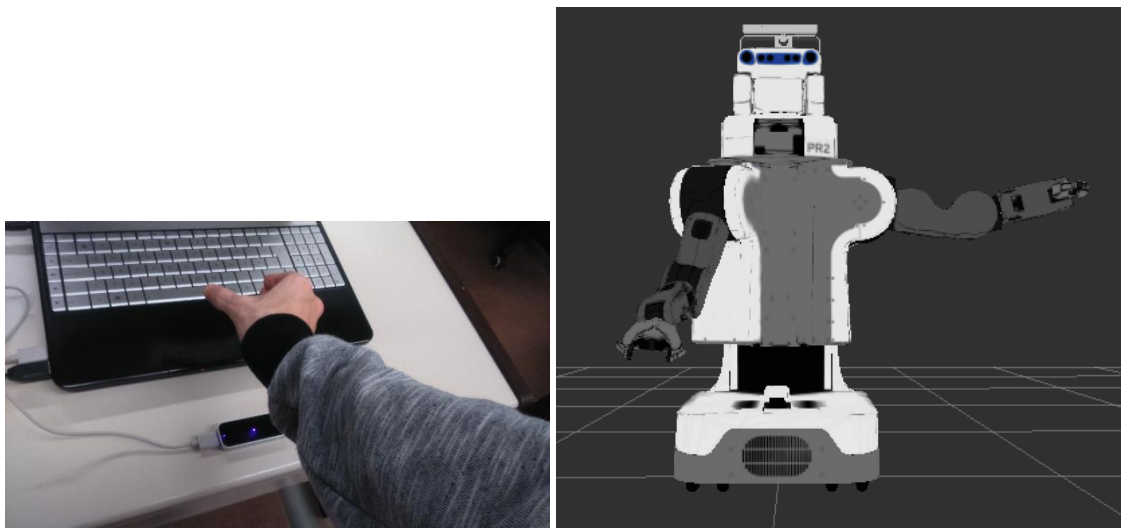

Figura 18: Controlo do robô - garra para a frente, a apontar para baixo e aberta na horizontal.

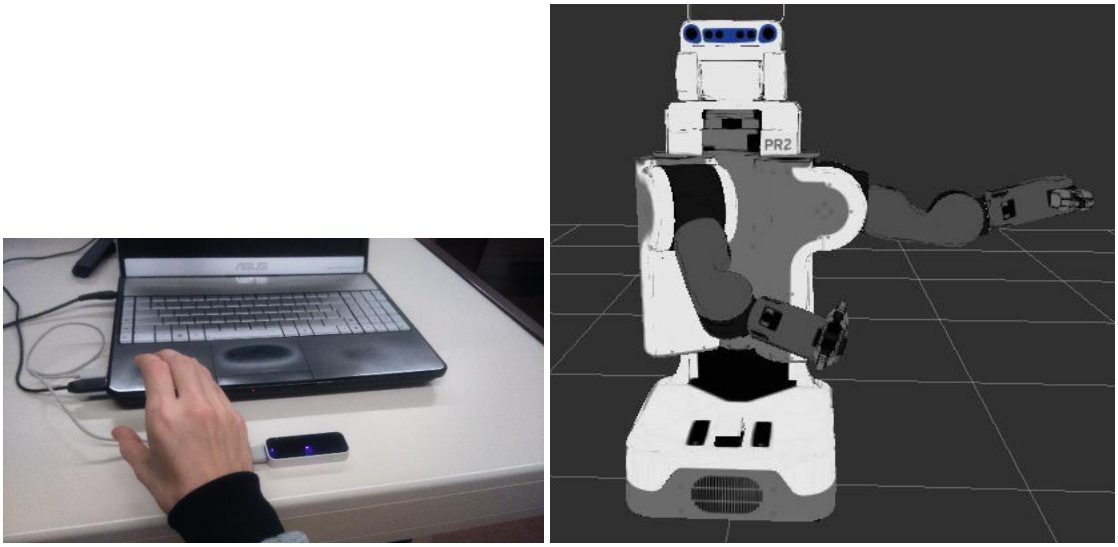

Figura 19: Controlo do robô - garra para a esquerda e para baixo, a apontar para a frente e aberta quase na vertical.

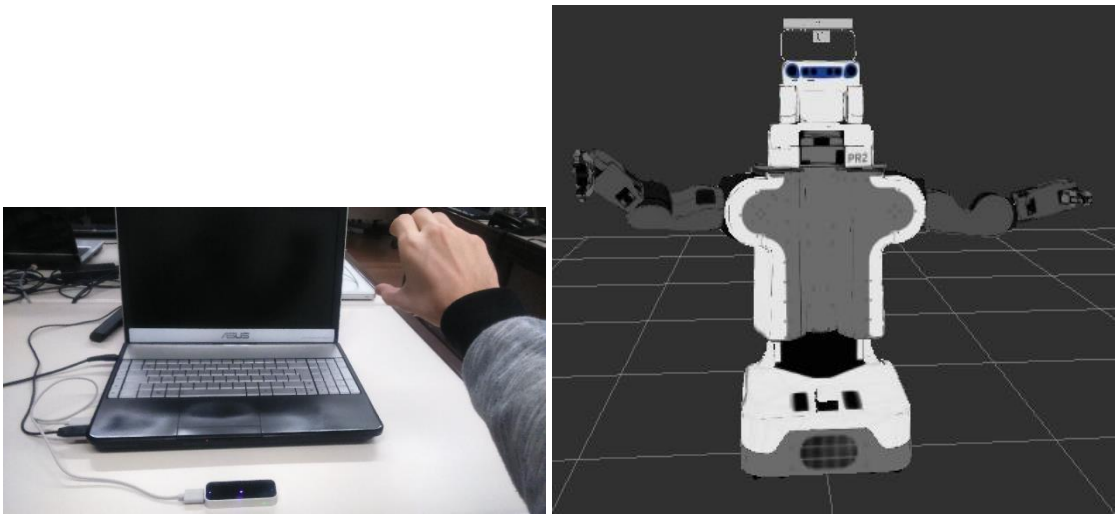

Figura 20: Controlo do robô - garra para a direita e para cima, a apontar para a frente e aberta na vertical.

Todos os testes foram feitos em simulação e o robô e as suas trajetórias foram visualizados com a ferramenta RViz, por não ser possível avançar para um robô real em tempo útil. Os testes feitos mostraram que o reposicionamento da garra nas três dimensões é intuitivo: o facto da posição da garra ser análoga à posição da mão do utilizador torna mais natural a interação num espaço tridimensional, em comparação com o controlo por *joysticks*.

Os gestos e o modo de utilização são fáceis de aprender: o controlo por imitação é inato e o *trigger* do modo de precisão – mão esquerda com punho fechado – é um gesto simples com uma taxa de reconhecimento superior a 90% em 15 tentativas (14 em 15).

Na maioria dos casos, o efeito do modo de precisão é dispensável no reposicionamento intencional da garra, mas a mitigação de movimentos involuntários mostrou-se crucial.

O atraso causado pelo envio de mensagens por *socket* é negligenciável numa rede local mas os pedidos de planeamento e execução das trajetórias são o *bottleneck* de todo o sistema. O cálculo destas trajetórias e o movimento do robô para as sequências de posições desejadas, com vários pedidos feitos num curto espaço de tempo, criam um atraso considerável quando acumulado. Este atraso pode ser de meros milissegundos, chegando mesmo a valores como 3 segundos. Isto reduz a sincronia das ordens e respetivas ações, essencialmente inviabilizando este método de controlo em computadores de baixo ou médio desempenho.

Para além disso e como era esperado, a precisão obtida com o sensor de movimento e reconhecimento de gestos não rivaliza a precisão dos *joysticks*. Isto não se deve tanto ao detalhe com que o Leap Motion, ou outro sensor, deteta a posição e orientação das mãos do utilizador mas deve-se sim à inabilidade de um ser humano manter a sua mão numa posição e orientação constantes durante vários segundos, enquanto se foca na tarefa que tenta realizar sem olhar para a sua mão. Uma potencial solução para este problema é a implementação de um gesto da mão secundária que ordena o robô a parar completamente e a ignorar futuros movimentos da mão que é imitada.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo, tiram-se conclusões relativas ao estudo e trabalho realizados, discute-se a satisfação dos objetivos e dão-se direções para trabalho futuro.

5.1 Satisfação dos objetivos

A análise feita às tecnologias existentes indica que o estado da arte das interfaces naturais por gestos evoluiu nos últimos anos de tal forma que a comercialização deste tipo de sistemas é já uma realidade. De modo similar, os *Head-Mounted Displays* e os ambientes imersivos que estes dispositivos permitem criar tornaram-se populares e um alvo de investigação e desenvolvimento. O controlo remoto de braços robóticos usando estas tecnologias está ainda por explorar na totalidade.

A visualização do ambiente onde se encontram os objetos com os quais interagir foi conseguida com duas câmaras 2D dispostas lado a lado a produzir *frames* que são processadas e passadas a um Oculus Rift, servindo de olhos ao utilizador. A perceção da profundidade e tridimensionalidade dos objetos, pelo menos para distâncias às câmaras inferiores a 2 metros, foi obtida de forma fiável.

A leitura da posição e orientação dos braços, mãos e dedos do utilizador foi concretizada usando um Leap Motion e o reconhecimento de gestos simbólicos foi implementado através de restrições aos movimentos detetados por este sensor. Isto foi conseguido com uma taxa de reconhecimento elevada.

O controlo do braço robótico por meio de imitação dos movimentos da mão do utilizador e por meio de gestos simbólicos, implementado sobre a *framework* ROS e o *software* MoveIt!, mostrou-se intuitivo, dada a correspondência direta entre os movimentos da mão imitada e os da garra robótica. Apesar disso, este sistema é menos preciso do que o controlo por *joysticks* e o

desfasamento temporal entre as ordens e as ações é problemático e deverá ser examinado no futuro.

5.2 Trabalho futuro

É interessante especializar o sistema e estudar as ações do robô para um género de tarefas em particular, e definir novos gestos da mão esquerda para os representar. Para além disso, outra funcionalidade útil é forçar o robô a parar por completo e a não imitar futuros movimentos da mão direita do utilizador, até indicado o contrário. Na implementação destes gestos adicionais, poderá ser necessário implementar um mecanismo de aprendizagem e treino de gestos, caso estes se tornem demasiado complexos para serem definidos apenas por restrições. Esse mecanismo poderá basear-se em redes neuronais, *Support Vector Machines* ou *Hidden Markov Models*.

O modo de precisão deve ser afinado depois de intensivos testes num cenário real, com um braço robótico e objetos reais. Similarmente, é importante reduzir o atraso entre as ordens dadas pelo utilizador e as ações do robô.

Outros sensores de movimento poderão ainda ser avaliados, em especial a pulseira Myo. Neste caso, para reconhecer movimentos e gestos das duas mãos do utilizador, serão necessários dois dispositivos, um em cada braço.

Referências

Bell, T.R., 2014. *High-quality, real-time 3D video visualization in head mounted displays*.

Bonansea, L., 2009. *3D Hand gesture recognition using a ZCam and an SVM-SMO classifier*.

Bourke, P., 1996. Cross Correlation. Available at: http://paulbourke.net/miscellaneous/correlate/.

Carvalho, F., 2014. *Shamanic Interface for computers and gaming platforms*.

Chaudhary, A. et al., 2011. Intelligent Approaches to interact with Machines using Hand Gesture Recognition in Natural way: A Survey. *International Journal of Computer Science & Engineering Survey*, 2(1), pp.122–133. Available at: http://www.airccse.org/journal/ijcses/papers/0211cses09.pdf.

Chen, F.-S., Fu, C.-M. & Huang, C.-L., 2003. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, 21(8), pp.745–758.

Crystal, A. & Ellington, B., 2004. Task analysis and human-computer interaction: approaches, techniques and levels of analysis. , (August), pp.1–9.

Ellis, L.U., 2008. *Perception and Displays for Teleoperated Robots*.

Goodine, C., 2014. Thalmic Labs Developer Forum / SDK Feedback and Support / Myo Developer FAQ. Available at: https://developer.thalmic.com/forums/topic/255/.

Hansard, M. et al., 2012. *Time of Flight Cameras: Principles, Methods and Applications*, Springer.

LaValle, S., 2013a. Help! My Cockpit Is Drifting Away | Oculus Rift - Virtual Reality Headset for 3D Gaming. Available at: https://www.oculus.com/blog/magnetometer/.

Referências

LaValle, S., 2013b. Sensor Fusion: Keeping It Simple | Oculus Rift - Virtual Reality Headset for 3D Gaming. Available at: https://www.oculus.com/blog/sensor-fusion-keeping-it-simple/.

Leap Motion, 2014a. API Overview - Leap Motion C# and Unity SDK v2.2 documentation. Available at: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html.

Leap Motion, 2014b. Leap Motion | 3D Motion and Gesture Control for PC & Mac. Available at: https://www.leapmotion.com/product.

Manning, C.D., Raghavan, P. & Schütze, H., 2008. Support vector machines and machine learning on documents. In *Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press, pp. 319–348.

Meisner, J. & Knies, R., 2013. Collaboration, expertise produce enhanced sensing in Xbox One - The Official Microsoft Blog. Available at: http://blogs.microsoft.com/blog/2013/10/02/collaboration-expertise-produce-enhanced-sensing-in-xbox-one/.

Microsoft, 2015a. Kinect for Windows features. Available at: http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx.

Microsoft, 2015b. Learn the basics about developing with Kinect. Available at: http://www.microsoft.com/en-us/kinectforwindows/develop/learn.aspx.

Murakami, K. & Taguchi, H., 1991. Gesture Recognition using Recurrent Neural Networks. In *CHI '91 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, pp. 237–242.

Murata, A., 2004. Effects of duration of immersion in a virtual environment on postural stability. *International Journal of Human-Computer Interaction*, 17(4), pp.463–477.

Neto, P., Pires, J.N. & Moreira, A.P., 2009. Accelerometer-Based Control of an Industrial Robotic Arm. In *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*. Toyama: IEEE, pp. 1192–1197.

Oculus VR, 2014. Oculus Developer Guide - SDK Version 0.4.4. , pp.0–62. Available at: http://static.oculus.com/sdk-downloads/documents/Oculus_Developer_Guide_0.4.4.pdf.

Oculus VR, 2015. The All New Oculus Rift Development Kit 2 (DK2) Virtual Reality Headset | Oculus Rift - Virtual Reality Headset for 3D Gaming. Available at: https://www.oculus.com/dk2/.

Open Source Robotics Foundation, 2015. ROS.org | About ROS. Available at: http://www.ros.org/about-ros/ [Accessed January 26, 2015].

Platt, J.C., 1998. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. , pp.1–21.

Referências

Quigley, M. et al., 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*. Available at: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

Rind, A. et al., 2014. User Tasks for Evaluation: Untangling the Terminology Throughout Visualization Design and Development.

Seibert, J., 2014. *An Exploratory Study on Virtual Reality Head Mounted Displays and Their Impact on Player Presence*.

Sucan, I.A. & Chitta, S., 2015. MoveIt! Available at: http://moveit.ros.org/ [Accessed April 12, 2015].

Thalmic Labs, 2014. Myo - Tech Specs. Available at: https://www.thalmic.com/en/myo/techspecs.

Vapnik, V. & Cortes, C., 1995. Support-Vector Networks. *Machine Learning*, 20(3), pp.273–297.

Vatavu, R.-D., Anthony, L. & Wobbrock, J.O., 2012. Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes. In *ICMI '12 Proceedings of the 14th ACM International Conference on Multimodal Interaction*. Santa Monica, California, USA: ACM, pp. 273–280.

Waldherr, S., Romero, R. & Thrun, S., 2000. A Gesture Based Interface for Human-Robot Interaction.pdf. *Autonomous Robots*, 9(2), pp.151–173.

Yin, Y. & Davis, R., 2010. Toward natural interaction in the real world: real-time gesture recognition. In *ICMI-MLMI '10 International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*. New York, NY, USA: ACM.

Zhang, X. et al., 2011. A Framework for Hand Gesture Recognition Based on Accelerometer and EMG Sensors. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 41(6), pp.1064–1076. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5735233 [Accessed January 18, 2015].