

Plataforma para auscultação autónoma

DANIEL FILIPE FERNANDES LOPES

novembro de 2022

Plataforma para auscultação autónoma

Daniel Filipe Fernandes Lopes



Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

2022

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Daniel Filipe Fernandes Lopes, Nº 1170685, 1170685@isep.ipp.pt
Orientação científica: Luis Filipe Martins Pinto Coelho, lfc@isep.ipp.pt
Coorientação científica: Manuel Fernando dos Santos Silva, mss@isep.ipp.pt



Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

7 de novembro de 2022

Agradecimentos

O fim do mestrado, assim como a conclusão da presente dissertação, marca o final de uma das fases mais importante da minha vida. Após cinco anos no Instituto Superior de Engenharia do Porto, deixo aqui os meus sinceros agradecimentos a todos os que me acompanharam nesta longa mas, satisfatória aprendizagem. Em primeiro lugar, à minha família por sempre me apoiarem e incentivarem a fazer mais e melhor e serem um dos pilares na minha vida. À minha namorada, pelo papel importante de assistente que desempenhou no decorrer da implementação no robô e por todo o amor, carinho, paciência e compreensão que tem comigo. Sem ti não era igual. Ao orientador desta dissertação o Professor Doutor Luís Filipe Coelho e ao coorientador Professor Doutor Manuel Silva pela orientação, disponibilidade e apoio que sempre demonstraram. Ao Professor Doutor António Ferreira da Silva e ao Departamento de Engenharia Mecânica por me terem fornecido completa liberdade na utilização do braço robótico e do laboratório onde o mesmo estava. A todos os meus amigos e colegas de faculdade que se disponibilizaram a participar na elaboração desta dissertação. E a todos que, direta ou indiretamente, me ajudaram neste percurso.

Resumo

O papel da tecnologia na indústria da medicina tem promovido grandes avanços e transformações profundas. Tornar os sistemas de saúde mais pro-ativos, modernos e eficazes é o principal foco desta revolução tecnológica. O surgimento da pandemia COVID-19, não só acelerou, como impulsionou este crescimento. Principalmente, no que diz respeito aos meios de diagnóstico, dado que, na área da saúde, quanto mais rápido se diagnosticar, mais rápido se pode tratar. No sentido de melhorar as decisões estratégicas através da obtenção de informação mais objetiva, foram criados modelos e propostas que pretendem oferecer serviços mais autónomos e ao mesmo tempo seguros e ágeis, sem comprometer a eficiência.

Esta dissertação tem como objetivo apresentar uma proposta de uma plataforma para realizar auscultação autónoma nas costas das pessoas, capaz de ajudar o profissional de saúde a realizar avaliações essenciais que contribuam para o bem-estar do utente.

A partir de informações externas do corpo obtidas através de um modelo de previsão, foi possível calcular os pontos de auscultação. Estes pontos de auscultação serão depois transformados em coordenadas espaciais e inseridos num braço robô colaborativo. Este braço robótico é responsável por realizar o contacto entre a superfície do estetoscópio e o corpo do paciente, nos vários pontos de auscultação desejados.

Esta ferramenta, ao ser utilizada nos locais adequados para o efeito, apresenta vantagens tais como, a automatização do processo de auscultação por parte do prestador de cuidados, disponibilizando este para tarefas mais urgentes, e o possível rastreio de doenças mais cedo.

Para avaliar a plataforma de auscultação autónoma desenvolvida, esta foi colocada em prática nas costas de seis pacientes. Com estes testes foi possível concluir que o braço robótico simula a colocação do estetoscópio nos pontos desejados de auscultação com sucesso, dependendo da estrutura física do paciente. Apesar disso, devem ser aplicadas melhorias nesta plataforma, principalmente, no que toca à estrutura da mesma.

Palavras-Chave: Auscultação Autónoma, Aprendizagem de Máquina, MobileNet, UNet, InceptionResNet, Adam, Inteligência Artificial, Robótica, Saúde, Visão por Computador.

Abstract

The role of technology in the medical industry has promoted major advances and profound transformations. Making healthcare systems more proactive, modern, and effective is the main focus of this technological revolution. The emergence of the COVID-19 pandemic has not only accelerated, but driven this growth. Especially with regard to diagnostic means, since in healthcare, the faster you can diagnose, the faster you can treat. In order to improve strategic decisions by obtaining more objective information, models and proposals have been created that intend to offer more autonomous and at the same time safer and more agile services, without compromising efficiency.

This dissertation aims to present a proposal for a platform to perform autonomous auscultation on people's backs, capable of helping the health professional to perform essential assessments that contribute to the user's well-being.

From external body information obtained through a prediction model, it was possible to calculate auscultation points. These auscultation points will then be transformed into spatial coordinates and inserted into a collaborative robot arm. This robotic arm is responsible for making contact between the stethoscope surface and the patient's body at the various auscultation points desired.

This tool, when used in the appropriate locations, has advantages such as automating the auscultation process for the caregiver, making the caregiver available for more urgent tasks, and possible earlier screening for diseases.

To evaluate the developed autonomous auscultation platform, it was put into practice on the back of six patients. With these tests it was possible to conclude that the robotic arm simulates the placement of the stethoscope in the desired auscultation points successfully, depending on the physical structure of the patient. Nevertheless, improvements should be made to this platform, especially with regard to its structure.

Keywords: Autonomous Auscultation, Machine Learning, MobileNet, UNet, InceptionResNet, Adam, Artificial Intelligence, Robotics, Healthcare, Computer Vision.

Índice

Agradecimentos	v
Índice	i
Índice de Figuras	v
Índice de Tabelas	ix
Acrónimos	xi
1 Introdução	1
1.1 Motivação	2
1.2 Formulação do Problema	3
1.3 Objetivos	4
1.4 Estrutura	5
2 Estado da Arte	7
2.1 Estetoscópio digital	7
2.1.1 TytoCare	8
2.1.2 3M™ Littmann® CORE	8
2.1.3 Thinklabs ONE	9
2.1.4 Estetoscópio de uso suave	10
2.2 Plataformas para realizar auscultação autónoma	10
2.3 Discussão do estado de arte	12
3 Fundamentos Teóricos	15
3.1 Aprendizagem de máquina	15
3.2 U-Net	17
3.2.1 Segmentação semântica	18
3.2.2 U-Net - arquitetura	18
3.3 MobileNetV1	20

3.3.1	MobileNetV2	21
3.3.1.1	Arquitetura MobileNetV2	22
3.3.2	MobileNetV2-Unet	23
3.4	Inception	24
3.4.1	Inception-ResNet-v2	25
3.4.2	Inception-ResNet-V2-UNet	26
3.5	Aumento dos dados	26
3.6	Otimizador Adam	28
3.7	Função de perda	32
3.7.1	<i>Dice loss</i>	32
3.8	Métricas de classificação	33
3.8.1	<i>Jaccard</i>	33
3.8.2	<i>Intersection over union</i>	34
3.8.3	<i>Precision</i>	34
3.8.4	<i>Recall</i>	35
3.8.5	<i>Accuracy</i>	35
3.8.6	<i>F1-Score</i>	35
3.9	Conclusão do capítulo 3	36
4	Hardware e Interfaces	37
4.1	<i>Hardware</i>	37
4.1.1	Manipulador robótico UR3e	37
4.1.2	Computador	38
4.1.3	Câmera OAK-D	39
4.2	<i>Interfaces</i>	40
4.2.1	URsim	41
4.2.2	TensorFlow	41
4.2.2.1	Keras	43
4.2.3	DepthAI	45
4.2.4	<i>Sockets</i>	48
4.2.5	RTDE	50
4.3	Conclusão do capítulo 4	51
5	Implementação	53
5.1	Avaliação da profundidade obtida pela câmera	53
5.2	Treino do modelo de previsão para identificar os pontos a auscultar . . .	59
5.2.1	Primeira abordagem	59
5.2.2	Segunda abordagem	63
5.3	Auscultação autónoma	65
5.3.1	Cálculo do <i>offset</i> do alinhamento da câmera com o braço robótico	66
5.3.2	Previsão dos pontos de referência	68
5.3.3	Cálculo dos pontos a auscultar	69

5.3.4	Cálculo das coordenadas espaciais dos pontos a auscultar	74
5.3.5	Realização da auscultação	74
5.4	Conclusão do capítulo 5	77
6	Resultados	79
6.1	Pacientes presentes no <i>Dataset</i>	79
6.2	Pacientes não presentes no <i>Dataset</i>	85
6.3	Conclusão do capítulo 6	90
7	Conclusão	91
	Referências Bibliográficas	93
A	MobileNetV1	99
A.1	Convolução com profundidade	99
A.2	Convolução pontual	100
A.3	Arquitetura MobileNetV1	101
B	MobileNetV2	103
B.1	Resíduos invertidos	103
B.2	Engarrafamentos lineares	104
B.3	ReLU6	104
C	Programas em Python	105

Índice de Figuras

1.1	Pontos para auscultação nas costas	4
2.1	TytoCare	8
2.2	3M™ Littmann® CORE	9
2.3	Thinklabs ONE	10
2.4	Estetoscópio de uso suave	11
2.5	Plataforma para auscultação robótica com câmera LiDAR	11
2.6	Sistema robótico capaz de realizar auscultação autônoma do coração e pulmão baseado no robô TRINA	12
3.1	Exemplo de segmentação semântica	18
3.2	Arquitetura U-Net	19
3.3	Exatidão da ImageNet Top-1 tendo em conta o número de <i>Multiply-Accumulates</i> (MAC)	20
3.4	Convoluções profundas separadas	21
3.5	Aplicações da MobileNet	21
3.6	Visão geral da arquitetura da MobileNetV2	22
3.7	Arquitetura MobileNetV2	23
3.8	MobileNetV2 melhora a velocidade (latência reduzida) e aumenta a precisão da ImageNet Top 1	23
3.9	Arquitetura MobileNetV2-UNet	24
3.10	Arquitetura da rede Inception-ResNet-v2	25
3.11	Características de algumas redes neuronais presentes no Keras	26
3.12	Aumento de dados	27
3.13	Otimizador Adam	30
3.14	Comparação do desempenho no custo do treino	32
3.15	<i>Sørensen–Dice coefficient</i>	33
3.16	Máscara objetivo e prevista	34
3.17	Interseção e união das máscaras	35
4.1	Arquitetura geral	38

4.2	Robô UR3e	38
4.3	Simulador URSim	42
4.4	Arquitetura da API DepthAI	45
4.5	Nó XLinkIn	46
4.6	Nó XLinkOut	47
4.7	<i>Pipeline</i> para obter a imagem RGB da câmara	48
4.8	Exemplo de utilização de <i>sockets</i>	49
5.1	<i>Pipeline</i> para obter a imagem de profundidade e outra de disparidade.	54
5.2	Teste com objeto alvo a 600 mm	54
5.3	<i>Boxplot</i> da coordenada z com o objeto a 600 mm	55
5.4	Teste com objeto a 700 mm	55
5.5	<i>Boxplot</i> da coordenada z com o objeto a 700 mm	56
5.6	Teste com objeto a 800 mm	56
5.7	<i>Boxplot</i> da coordenada z com o objeto a 800 mm	56
5.8	Teste com objeto alvo a 700 mm e objeto idêntico a 600 mm	57
5.9	<i>Boxplot</i> da coordenada z do objeto alvo a 700 mm e com um objeto idêntico a 600 mm	57
5.10	Teste com objeto alvo e objeto idêntico a 700 mm	57
5.11	<i>Boxplot</i> da coordenada z do objeto alvo a 700 mm e com um objeto idêntico a 700 mm	58
5.12	Teste com objeto alvo a 700 mm e objeto idêntico a 800 mm	58
5.13	<i>Boxplot</i> da coordenada z do objeto alvo a 700 mm e com um objeto idêntico a 800 mm	59
5.14	Exemplo de uma imagem de costas e a sua respetiva máscara, presentes no <i>Dataset</i>	60
5.15	Técnicas de aumento de dados da primeira abordagem	61
5.16	Exemplo de um par de imagens com os pontos de referência para treinar	64
5.17	Técnicas utilizadas para o aumento dos dados	64
5.18	Arquitetura geral do projeto.	65
5.19	Cálculo do <i>offset</i> do alinhamento do braço robótico com a câmara	67
5.20	Cálculo das coordenadas espaciais do <i>offset</i>	67
5.21	<i>Frame</i> das costas da pessoa	68
5.22	Extração de fundo do <i>frame</i> das costas da pessoa	69
5.23	Pontos de referência nas costas da pessoa	69
5.24	Pontos de auscultação 2D	71
5.25	Primeira parte do algoritmo de cálculo dos pontos 2D de auscultação	73
5.26	Segunda parte do algoritmo de cálculo dos pontos 2D de auscultação	74
5.27	Coordenadas espaciais 3D dos pontos a auscultar	75
5.28	Trajeto realizado pelo braço robótico	76
5.29	Referencial do braço robótico	76
5.30	Algoritmo de controlo do braço robótico	78

6.1	<i>Frames</i> das costas dos pacientes utilizadas para a previsão dos pontos de referência	80
6.2	Extração do fundo das <i>frames</i> das costas dos pacientes	80
6.3	Pontos de referência previstos nas costas dos quatro pacientes	81
6.4	Pontos de auscultação calculados nos quatro pacientes	81
6.5	Coordenadas espaciais 3D dos pontos de auscultação dos quatros pacientes	82
6.6	Braço robótico nos pontos de auscultação do paciente 1	82
6.7	Braço robótico nos pontos de auscultação do paciente 2	83
6.8	Braço robótico nos pontos de auscultação do paciente 3	83
6.9	Braço robótico nos pontos de auscultação do paciente 4	84
6.10	Pontos auscultados nos quatro pacientes	85
6.11	<i>Frames</i> das costas dos pacientes não presentes no <i>Dataset</i> utilizadas para a previsão dos pontos de referência	86
6.12	Extração do fundo das <i>frames</i> das costas dos pacientes não presentes no <i>Dataset</i>	86
6.13	Pontos de referência previstos nas costas dos dois pacientes	86
6.14	Pontos de auscultação calculados nos dois pacientes	87
6.15	Coordenadas espaciais 3D dos pontos de auscultação dos dois pacientes . .	87
6.16	Braço robótico nos pontos de auscultação do paciente 5	88
6.17	Braço robótico nos pontos de auscultação do paciente 6	88
6.18	Colisão da junta <i>elbow</i> do braço robótico com a bancada onde se encontra instalado, ao tentar alcançar o ponto <i>L4</i> do paciente 6	89
6.19	Pontos auscultados nos dois pacientes não presentes no <i>Dataset</i>	90
A.1	Convolução normal	99
A.2	Convolução profunda	100
A.3	Convolução pontual	100
A.4	Na esquerda, camada convolucional normal com <i>batch normalization</i> e ReLU. Na direita, convolução profunda separável com camadas em profundidade e em pontos, seguida de <i>batch normalization</i> e ReLU	101
A.5	Arquitetura MobileNetV1	102
B.1	Um bloco residual liga largas camadas com uma <i>skip connection</i> enquanto as camadas intermédias são estreitas	103
B.2	Um bloco residual invertido liga camadas estreitas com uma <i>skip connection</i> enquanto as camadas intermédias são largas	104

Índice de Tabelas

4.1	Especificações principais do manipulador robótico UR3e	39
4.2	Especificações do computador utilizado	39
4.3	Especificações da câmera OAK-D	40
4.4	Diferenças entre os protocolos TCP e UDP	49
5.1	Mudanças no treino do <i>Dataset</i> e no respetivo <i>Dataset</i>	60
5.2	Resultados do modelo utilizado para a segmentação de imagem em humanos quando aplicado no <i>Dataset</i> de validação/teste em que foi treinado	61
5.3	Resultados do treino, em concordância com a Tabela 5.2	63
5.4	Caraterísticas do <i>Dataset</i> utilizado para o treino da rede	65
5.5	Resultados do treino, em concordância com a Tabela 5.4	65

Acrónimos

Acrónimo	Descrição	Página
AdaGrad	<i>Adaptive Gradient Algorithm</i>	29
ADAM	<i>Adaptive Moment Estimation</i>	28
API	<i>Application Programming Interface</i>	40
AR	<i>Augmented Reality</i>	40
BO	<i>Bayesian Optimization</i>	12
BSD	<i>Unix Berkeley Software Distribution</i>	48
CPU	<i>Central Processing Unit</i>	42
CNN	<i>Convolutional Neural Network</i>	21
E/S	Entradas/Saídas	49
FCN	<i>Fully Convolutional Networks</i>	18
GPU	<i>Graphics Processing Unit</i>	42
HSV	<i>Hue, Saturation, Value</i>	67
IA	Inteligência Artificial	39
iOS	<i>iPhone Operating System</i>	09
IoU	<i>Intersection over Union</i>	34
IP	<i>Internet Protocol</i>	40
ISEP	Instituto Superior de Engenharia do Porto	i
LiDAR	<i>Light Detection and Ranging</i>	10
MAC	<i>Multiply-Accumulates</i>	20
NASA	National Aeronautics and Space Administration	43
OAK	<i>OpenCV AI Kit</i>	39
PC	<i>Personal Computer</i>	09
POI	<i>Point-Of-Interest</i>	54
RGB	<i>Red-Green-Blue</i>	39
RMSP	<i>Root Mean Square Propagation</i>	28
ROI	<i>Region-Of-Interest</i>	54
RPi	<i>Remote Programming Interface</i>	45
RSN	Relação Sinal-Ruído	10
RTDE	<i>Real-Time Data Exchange</i>	40
SoM	<i>System on Module</i>	39
TCP	<i>Transmission Control Protocol</i>	48
TPU	<i>Tensor Processing Unit</i>	42

Acrónimo	Descrição	Página
UDP	<i>User Datagram Protocol</i>	48
UPV	Unidade de Processamento Visual	39
UR	Universal Robot	50
USB	<i>Universal Serial Bus</i>	38
VR	<i>Virtual Reality</i>	40

Capítulo 1

Introdução

A automatização foi introduzida em quase todos os aspetos da vida humana, desde robôs industriais a ferramentas que nos simplificam a vida. Utilizada para reduzir a mão-de-obra, aumentar a eficiência e baixar os custos, está em todos os lugares do nosso dia-a-dia, mesmo que disso não nos apercebamos. O termo automação começou a ser utilizado inicialmente nos anos 40, quando a empresa de motores Ford introduziu um departamento de automação [1]. À medida que os avanços nos sensores se foram sucedendo, a automatização foi também ganhando uma maior base a nível global e com os avanços nos processadores, mais oportunidades surgiram para o seu desenvolvimento e crescimento.

Criados para desempenhar tarefas, normalmente, realizadas por seres humanos, os robôs colaborativos transformaram o universo da robótica tradicional e dos meios de produção. Introduzidos nas fábricas, pela primeira vez, nos anos 80, os sistemas robóticas ganharam o seu destaque no mundo de produção, principalmente no que diz respeito à indústria automóvel [1].

Na indústria médica, a introdução da robótica e das tecnologias autónomas contribuíram e foram fundamentais para enfrentar a sobrecarga do número de pacientes por profissional de saúde e manter o nível de qualidade existente nos serviços de saúde. Neste contexto, a automação permite, entre outras possibilidades, levar a cabo atividades de diagnóstico mais fáceis e mais seguras, não implicando, necessariamente, a presença e contacto direto entre um médico e o seu utente.

A auscultação tem sido uma componente integral do exame clínico desde o século XIX e é um método de rastreio altamente rentável para a deteção de sintomas clínicos anormais [2]. A auscultação continuará a ser essencial nos tempos modernos dado que a doença cardiopulmonar é uma das maiores causas subjacentes e diretas de morte a nível mundial, com uma grande influência na qualidade de vida e despesas de saúde [3]. Além disso, a auscultação é uma técnica de diagnóstico viável para doentes com COVID-19 e

pode ser utilizada como instrumento de acompanhamento para doentes não críticos com COVID-19. A auscultação remota é vantajosa em termos de prevenção de infeções, uma vez que a auscultação convencional requer que os clínicos estabeleçam contacto físico com os pacientes.

Os estetoscópios eletrónicos surgiram recentemente como soluções interessantes para auscultações remotas [4]. O estetoscópio eletrónico permite a visualização de sonogramas do coração e dos pulmões durante a auscultação, tornando mais simples a distinção entre vários tipos de sons cardíacos e pulmonares, bem como o diagnóstico assistido por computador na doença arterial coronária [5]. O desafio na auscultação é que a sua eficácia depende fortemente das capacidades auditivas e da perícia dos médicos. A inteligência artificial está a ser aplicada aos estetoscópios para diminuir a subjetividade da auscultação [6]. Embora a tecnologia tenha aumentado a eficácia da auscultação, a prática ainda envolve médicos ou pessoal de saúde para colocar estetoscópios portáteis no corpo do paciente, exigindo a sua presença física.

Além disso, durante a auscultação, devem colocar o estetoscópio com forças de contacto adequadas contra a pele do paciente, ou seja, para garantir a segurança do paciente e para remover ruídos estranhos. Dado a escassez de profissionais de saúde, particularmente nas zonas rurais onde os recursos médicos são escassos [5], a criação de uma plataforma robótica que possa realizar auscultação autónoma com elevada precisão e repetibilidade sem depender da intervenção do médico seria útil.

O contributo deste documento é criar uma plataforma robótica que permita a colocação autónoma do estetoscópio com base em informações externas do corpo, ao mesmo tempo que satisfaz a segurança do paciente em termos de contacto da superfície do estetoscópio com o corpo.

1.1 Motivação

Na prevenção, diagnóstico e tratamento de doenças e outras condições médicas, os principais avanços da medicina procedem das várias inovações tecnológicas.

A tecnologia e a sua influência na ciência permitem encontrar soluções que sejam adequadas tanto para os profissionais de saúde como para os seus pacientes.

No que diz respeito aos equipamentos para diagnóstico, disponibilizar um meio que seja capaz de observar, diagnosticar e perceber a doença, como é o caso de um braço robótico para auscultar pessoas, seria um contributo capaz de revolucionar a forma como os cuidados de saúde funcionam, devido à sua agilidade em realizar esta tarefa em cada utente, através também de um *software* que permite a gravação e, posteriormente, a sua reprodução em qualquer lugar e momento, ou até mesmo a sua análise final.

Disponibilizar tal ferramenta, que contribui para o melhoramento de fluxos hospitalares, por exemplo, e contribuir para a área, é a motivação deste projeto.

1.2 Formulação do Problema

Auscultação é um termo médico utilizado para ouvir os sons do coração, pulmões, artérias e estômago (abdómen), onde o prestador de cuidados de saúde normalmente utiliza um estetoscópio. O estetoscópio é colocado diretamente na área a auscultar e tem como função ouvir vários sons, qualidade, volume, tempo e outros fatores. A auscultação é um método simples e seguro de detecção de potenciais condições médicas, não é um método invasivo e é uma das técnicas mais antigas utilizadas atualmente pelos prestadores de cuidados de saúde [7].

Um estetoscópio é um dispositivo que ouve os sons do corpo e é composto por uma campânula e um diafragma. A campânula e o diafragma são utilizados pelo prestador de cuidados de saúde para detetar os sons graves e agudos, respetivamente. Os tubos de borracha ligam a campânula e o diafragma aos auriculares que o prestador de cuidados de saúde insere nos seus ouvidos [7].

Uma auscultação pulmonar é um procedimento em que o prestador de cuidados de saúde ouve os sons que os pulmões fazem. Normalmente eles dizem ao paciente para se sentar direito e serão necessárias várias respirações profundas através da boca, para ouvir os pulmões. O prestador de cuidados de saúde examina primeiro o peito com o estetoscópio, onde este é movido por vários pontos de auscultação no tórax. A auscultação normalmente começa no topo dos pulmões e vai até ao fundo da caixa torácica, onde ouve cada pulmão e compara os sons. O estetoscópio depois é colocado nas costas e começa de novo de cima para baixo de um lado para o outro. Em cada local de auscultação, ele ouve um ciclo respiratório completo. Por fim, ele compara os sons que ouve na frente e atrás dos pulmões [7].

O prestador de cuidados de saúde ouve [7]:

- duração: mede a duração das inalações e exalações;
- frequência: mede as ondas sonoras e as vibrações;
- amplitude: nível de ruído ou intensidade nas respirações;
- qualidade: quaisquer características distintivas ou sons anormais, tais como chiado.

Os diferentes pontos de auscultação dos pulmões produzem sons diferentes. Mas tipicamente os sons normais dos pulmões são suaves e ocos e podem ser ouvidos claramente quando se inspira [7]. O prestador de cuidados de saúde pode ouvir vários sons anormais, como, por exemplo, assobios agudos, roncos e chiados [7].

Neste projeto foi criada uma plataforma para realizar apenas a auscultação autónoma nas costas. Para isso, foram utilizados os pontos de auscultação demonstrados na

Figura 1.1, onde se teve apenas em atenção as posições das omoplatas para não auscultar em cima das mesmas, devido a estas dificultarem a auscultação (bloqueia a transmissão do som). Estes pontos são obtidos através do conhecimento da anatomia Humana e de pontos de referência cuja localização é estimada através de um sistema de visão baseado em inteligência artificial. [7].

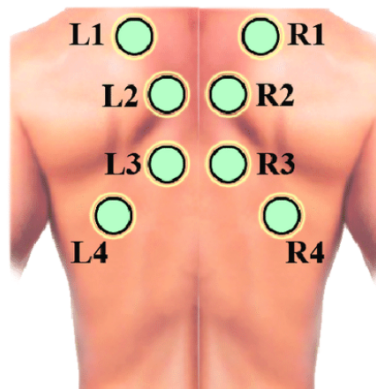


Figura 1.1: Pontos para auscultação nas costas [8]

1.3 Objetivos

O objetivo principal desta dissertação é definir uma solução apropriada para auscultação remota. A solução proposta tem como alvo a auscultação das costas das pessoas. Este sistema deve:

- treinar um modelo de previsão;
- calcular o desalinhamento do braço robótico e da câmera (se houver);
- visualizar as costas da pessoa;
- identificar os pontos referência no corpo Humano;
- através destes pontos anteriores calcular os pontos de auscultação;
- calcular a localização espacial dos pontos de auscultação em relação à câmera;
- levar o braço robótico aos pontos de auscultação espaciais;
- aproximar o robô até tocar na pessoa, monitorizando em tempo real a força exercida pelo mesmo;
- realizar a auscultação.

Este deve também realizar a auscultação autónoma com uma elevada precisão e repetibilidade sem depender da intervenção de um médico.

1.4 Estrutura

Esta dissertação encontra-se organizada em sete capítulos. No segundo capítulo encontra-se o Estado da Arte, que endereça um estudo sobre as soluções existentes de estetoscópios digitais presentes no mercado, e alguns dos estudos e desenvolvimentos já realizados na criação de plataformas de auscultação autónoma.

A fundamentação teórica, que dá suporte aos desenvolvimentos deste trabalho, é descrita no Capítulo 3. Neste é realizado uma introdução à aprendizagem de máquinas, às redes utilizadas para treinar o modelo de previsão, e às ferramentas utilizadas para treinar este mesmo.

O *Hardware* e as Interfaces são descritas no Capítulo 4, neste consta toda a estrutura e arquitetura do sistema desenvolvido, e as interfaces utilizadas.

O Capítulo 5 apresenta os desenvolvimentos realizados, as duas abordagens para a criação do modelo de previsão, a utilização deste, o cálculo dos pontos a auscultar, e a auscultação em si.

A plataforma foi testada nas costas de seis pacientes e os resultados estão presentes no Capítulo 6.

Por fim a conclusão, no Capítulo 7, tece comentários a todo o trabalho realizado e resume os desenvolvimentos futuros.

Capítulo 2

Estado da Arte

A auscultação pulmonar tem sido uma das principais características nos exames físicos regulares ao longo dos anos e é um exame não-invasivo muito útil para diagnóstico de doenças pulmonares. Embora o papel do estetoscópio na auscultação pulmonar esteja a ser desafiado pelas tecnologias mais recentes, o instrumento continua a ser uma peça fundamental nos instrumentos de diagnóstico para o médico e o instrumento médico de diagnóstico mais eficiente utilizado para fornecer pistas clínicas relativas à saúde do paciente. Abaixo vão ser apresentados alguns dos mais recentes desenvolvimentos no estetoscópio e os estudos já realizados na criação de plataformas para auscultação autónoma.

2.1 Estetoscópio digital

O desenvolvimento dos estetoscópios eletrônicos e digitais deu início a uma nova geração de auscultação assistida por computador. Estes são compostos por três diferentes módulos [2]:

- aquisição de dados;
- pré-processamento;
- processamento de sinais.

O estetoscópio digital transforma sons acústicos em sinais eletrônicos, que podem ser ainda mais amplificados para otimizar a auscultação. A digitalização subsequente dos sinais eletrônicos permite a transmissão de sons cardíacos para um computador, para análise automática, visualização gráfica e armazenamento [2].

Além disso, vários estetoscópios digitais podem, através de ligação Bluetooth, transmitir sons cardíacos sem fios para uma rede de processamento remoto, promovendo a evolução e potenciais aplicações de telemedicina [2].

Estudos recentes sugeriram a viabilidade, na fase pré-hospitalar, de uma auscultação cardíaca com uma boa precisão diagnóstica utilizando *smartphones*, embora as aplicações utilizadas necessitem de melhorias [2].

O estetoscópio digital suprime o ruído ambiente, o que permite ao médico auscultar os sons cardíacos e pulmonares o mais fielmente possível ao original. Isto proporciona um diagnóstico mais preciso com base numa avaliação clínica quantificável e promove assim melhores cuidados médicos. A possibilidade de interpretação acústica automática abre cenários futuros intrigantes não só no diagnóstico cardiovascular, mas também na melhoria do ensino clínico à beira do leito. Finalmente, os médicos com perda auditiva podem também beneficiar de estetoscópios eletrónicos [2].

Abaixo descrevem-se ver alguns produtos presentes no mercado de estetoscópio digitais, e dois estudos realizados na mesma área.

2.1.1 TytoCare

O sistema TytoCare permite o exame físico remoto de pacientes com COVID-19. Este sistema é uma plataforma digital concebida para exames físicos à distância, incluindo um estetoscópio para auscultação do coração e pulmões, um otoscópio digital para visualização da membrana timpânica, um termómetro digital, um oxímetro digital e um depressor de língua que permite o exame visual da faringe. Os dados são transmitidos visualmente, gravados e ainda está também disponível a realização de videoconferência. A Figura 2.1 mostra os componentes do sistema, um monitor digital que apresenta instruções de auto-utilização (caso o paciente coloque o dispositivo para o uso de um médico à distância), e os vários componentes acima mencionados [9].



Figura 2.1: TytoCare [9]

2.1.2 3M™ Littmann® CORE

A acústica deste estetoscópio cardiológico contém uma amplificação até 40x, o que ajuda na auscultação de sons subtis, e com a tecnologia de cancelamento ativo de ruído

ajuda a eliminar os sons perturbadores comuns presentes nos ambientes agitados. Este também possui um botão que permite comutar rapidamente entre os modos de audição analógica e amplificada, segundo as necessidades. Além disso, os dois diafragmas sintonizáveis (adulto e pediátrico) permitem auscultar sons de alta e baixa frequência, sem reposicionar o auscultador.

Com a conectividade sem fios (Bluetooth), que confere a capacidade de emparelhar com diversos dispositivos móveis, funciona com *software* Eko em dispositivos iPhone *Operating System* (iOS) e Android, que permite guardar e anotar gravações de 15, 30, 60 ou 120 segundos num painel seguro. É também possível criar uma biblioteca de sons cardíacos para monitorizar a evolução de uma doença, para fins educativos, ou para partilhar as gravações com os colegas de forma segura. E ainda é possível utilizar a inteligência artificial Eko para ajudar na deteção de sopros cardíacos [10]. A Figura 2.2 mostra o estetoscópio 3M™ Littmann® CORE.



Figura 2.2: 3M™ Littmann® CORE [11]

2.1.3 Thinklabs ONE

Este estetoscópio eletrónico alterou a imagem tradicional dos estetoscópios devido a não possuir haste ou olivas e ao seu *design* prático e fácil de transportar, em vez disso, trata-se apenas de um pequeno dispositivo circular. Este estetoscópio vem com um par de auscultadores com entrada *jack* de 3,5 mm. O diafragma deste estetoscópio é muito sensível à pressão e permite ouvir os sons de baixa frequência aplicando apenas um pouco de pressão. A sensibilidade pode ser ajustada através do ecrã presente no mesmo, sendo este ecrã fácil de operar. Este estetoscópio electrónico não tem um sistema de cancelamento de ruído, por conseguinte, pode ser difícil utilizar este modelo num fundo perturbador. A auscultação pode ser partilhada com dispositivos como iOS, Mac, *Personal Computer* (PC), ou Android, sendo que a característica da conectividade sem fios não está disponível, mas em troca, inclui um adaptador Thinklink [12]. A Figura 2.3 mostra o estetoscópio Thinklabs ONE conectado a um auscultador.



Figura 2.3: Thinklabs ONE [13]

2.1.4 Estetoscópio de uso suave

Neste estudo foi utilizada uma classe de tecnologias que oferecem auscultação contínua em tempo real e sem fios, utilizando um estetoscópio com um sistema de toque macio como ferramenta de diagnóstico para várias doenças. Neste trabalho [14] foi construído um estetoscópio com um sistema de uso suave para auscultação cardiopulmonar ambulatorial utilizando tecnologias com eletrônica avançada, mecânica flexível, e material suave, que servem como um novo sistema de uso auto-operável para monitorização cardiovascular e respiratória contínua.

O objetivo deste trabalho foi a concepção de um dispositivo para recolha de dados cardio respiratórios durante as atividades diárias, para diagnosticar várias anomalias pulmonares. Neste trabalho, são estudados a melhoria da Relação Sinal-Ruído (RSR) a partir da recolha de sons denotados por ondas, a minimização dos circuitos para tornar o modelo do dispositivo mais compacto, e o treino de um modelo de aprendizagem máquina para identificar corretamente vários problemas pulmonares. Foi também desenvolvida uma aplicação de fácil utilização para o dispositivo móvel que pode gravar sons cardíacos e pulmonares, seguir e exibir sinais em tempo real, diagnosticar automaticamente vários sons pulmonares anormais, e carregar informação para uma memória local sincronizada de forma remota e segura [14]. A Figura 2.4 demonstra este estetoscópio e a sua interface.

2.2 Plataformas para realizar auscultação autónoma

No trabalho desenvolvido em [15], demonstrado na Figura 2.5, o objetivo foi desenvolver uma plataforma de auscultação robótica onde fosse possível estimar as posições de auscultação e colocar com segurança o estetoscópio na posição estimada. O método de estimativa baseia-se no registo com uma câmara *Light Detection and Ranging* (LiDAR). Para realizar a auscultação totalmente autónoma, o sistema robótico reconhece

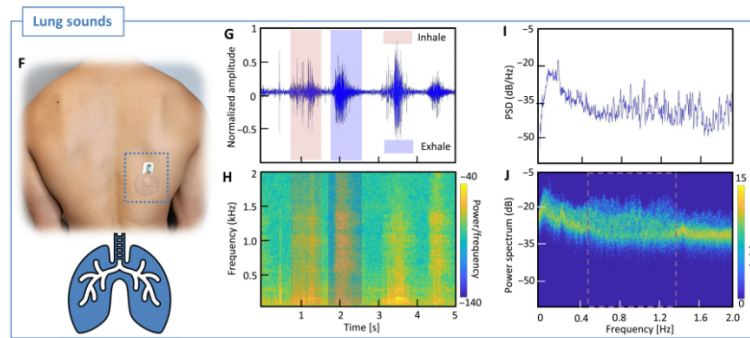


Figura 2.4: Estetoscópio de uso suave [14]

as posições de auscultação com base na forma do corpo inteiro e coloca o estetoscópio nas posições de auscultação determinadas, considerando a relação posicional entre o corpo e o estetoscópio. Neste estudo, a câmara LiDAR é utilizada para reconstruir e adquirir o contorno de toda a superfície do corpo, assim como combinar as informações de profundidade adquiridas sequencialmente como dados das nuvens de pontos. Com a informação reconstruída sobre a forma do corpo, as posições de auscultação são estimadas com base nas características anatômicas da superfície do corpo. Para colocar o estetoscópio em segurança sobre a superfície do corpo, foi desenvolvido um vetor final capaz de aplicar uma força de contacto constante contra a superfície do corpo com um mecanismo passivo único. O mecanismo passivo compreende uma mola linear, um servomotor linear, e um sensor ótico de distância. Este pode gerar diferentes forças constantes de contacto quando a quantidade de compressão da mola linear é controlada dentro de um determinado alcance em tempo real [15].

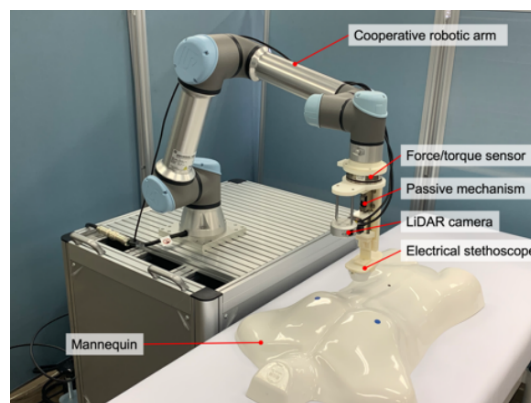


Figura 2.5: Plataforma para auscultação robótica com câmera LiDAR [15]

O trabalho desenvolvido em [16], ilustrado na Figura 2.6 demonstra o primeiro sistema robótico capaz de realizar auscultação autônoma do coração e pulmão, baseado no robô TRINA mostrado também na mesma figura. O objetivo do robô é fornecer sons de

utilidade diagnóstica, com alta qualidade, para um médico ouvir, em vez de realizar o diagnóstico em si. Em primeiro lugar, o sistema capta um *scan* da nuvem de pontos 3D do paciente, regista um modelo do corpo humano, estima a localização de pontos-chave anatômicos, e produz um mapa prévio de locais de auscultação de alta qualidade. Em seguida, adota um planeamento informativo do caminho, utilizando *feedback* sonoro para procurar de forma adaptável a região de interesse para uma auscultação de alta qualidade. O *feedback* auditivo baseia-se em estimadores de qualidade sonora treinados numa base de dados de gravações de estetoscópios cardíaco e pulmonar. Para determinar o local ideal de auscultação, foi formulado um problema de *Bayesian Optimization* (BO) onde o campo de qualidade sonora desconhecida foi estimado como um modelo de processo gaussiano residual semi-paramétrico, com um mapa prévio que depende de um *offset* de tradução latente e parâmetros de escala de qualidade sonora. Por fim, neste trabalho foram realizadas experiências em quatro sujeitos humanos masculinos saudáveis, onde demonstram que o sistema executa a auscultação do coração e pulmão automaticamente, e localiza sons de qualidade de diagnóstico e com um tempo de execução semelhante ao da tele-operação por um humano treinado em auscultação na escola médica, e treinado para usar o robô pelos investigadores [16].

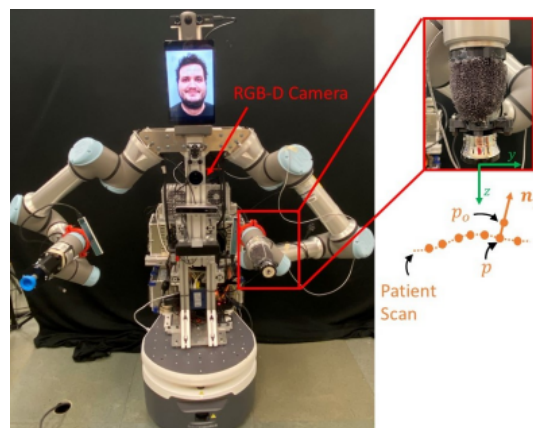


Figura 2.6: Sistema robótico capaz de realizar auscultação autónoma do coração e pulmão baseado no robô TRINA [16]

2.3 Discussão do estado de arte

Pelos estudos acima descritos, é possível verificar que existem poucos trabalhos realizados sobre plataformas para auscultação autónoma. Pode-se ver também que os robôs utilizados nesses estudos são braços robóticos colaborativos, até da mesma empresa, devido à segurança que eles fornecem. Aliado a este braço robótico está sempre um sensor/câmera para visualizar o paciente e a localização onde este mesmo braço robótico terá de ir.

Quanto aos estetoscópios, existe uma grande variedade, sendo a mais recente os estetoscópios digitais e eletrônicos, onde existe um constante trabalho em desenvolvimento. Estes são cada vez mais práticos e simples, permitindo uma fácil utilização. Possuem também várias características inovadoras tais como, cancelamento de ruído, amplificação digital e interfaces para visualizar os dados recebidos do paciente.

Os estetoscópios digitais poderão ser aliados à plataforma de auscultação autónoma para criar um equipamento mais completo, sendo a plataforma responsável por identificar os pontos de auscultação, e por efetuar o movimento do estetoscópio ao encontro do ponto no paciente, e o estetoscópio digital responsável por ouvir o som dos pulmões/coração e gravar na sua interface, ou até mesmo dar o resultado final, visto que alguns estetoscópios digitais possuem inteligência artificial para detetar se existe alguma anomalia.

Capítulo 3

Fundamentos Teóricos

Neste capítulo são abordados os conceitos teóricos utilizados para a criação deste projeto. É feita uma pequena introdução à aprendizagem de máquinas e à aprendizagem profunda, uma vez que este trabalho se enquadra nesse tema, são descritas as redes neurais utilizadas para prever os pontos relevantes nas costas das pessoas, as métricas utilizadas para avaliar o treino e o teste destas mesmas redes no Dataset criado, a função de perda e o otimizador utilizados para treinar estas mesmas redes e as técnicas existentes para aumentar os dados presentes no Dataset.

3.1 Aprendizagem de máquina

Aprendizagem de máquina é um subcampo da inteligência artificial, que é amplamente definido como a capacidade de uma máquina imitar um comportamento inteligente humano. Os sistemas de inteligência artificial são utilizados para executar tarefas complexas de uma forma semelhante aos humanos [17]. Isto significa que as máquinas podem reconhecer uma cena visual, compreender um texto escrito em linguagem natural, ou executar uma ação no mundo físico [17].

A aprendizagem de máquina utiliza dados, ou seja, números, fotos, ou textos, tais como transações bancárias, imagens de pessoas ou mesmo artigos de padaria, registros de reparação, dados de sensores, ou relatórios de vendas. Estes são recolhidos e preparados para serem utilizados como dados de treino, ou como informação sobre a qual o modelo de aprendizagem de máquina será treinado. Quanto mais dados, melhor será o programa [17].

De seguida, os programadores escolhem um modelo de aprendizagem de máquina para utilizar, fornecem os dados, e deixam o modelo treinar-se a si próprio para encontrar padrões ou fazer previsões. Com o tempo, o programador também poderá ajustar o

modelo, incluindo alterar os seus parâmetros, para o ajudar a conduzir para resultados mais precisos [17].

Alguns dados são retirados dos dados de treino para serem utilizados como dados de avaliação, o que testa a precisão do modelo de aprendizagem de máquina quando são utilizados novos dados. O resultado é um modelo que pode ser utilizado no futuro com diferentes conjuntos de dados [17].

Os algoritmos de aprendizagem de máquina que têm sucesso podem fazer diferentes coisas. A função de um sistema de aprendizagem de máquina pode ser **descritiva**, significando que o sistema usa os dados para explicar o que aconteceu; **preditiva**, significando que o sistema usa os dados para prever o que vai acontecer, como é o caso deste trabalho; ou **prescritiva**, significando que o sistema vai usar os dados para fazer sugestões sobre que ação tomar [17].

Em aprendizagem de máquina existem três subcategorias:

Os modelos **supervisionados** de aprendizagem de máquina que são treinados com conjuntos de dados etiquetados, que permitem que os modelos aprendam e cresçam com mais precisão ao longo do tempo. Por exemplo, um algoritmo pode ser treinado com imagens de cães e outras coisas, todas etiquetadas por humanos, e a máquina iria aprender formas de identificar imagens de cães por si só. A aprendizagem supervisionada é o tipo mais comum utilizado atualmente [17] e foi utilizada neste projeto.

Nos modelos **não supervisionados** de aprendizagem de máquina, o programa procura padrões em dados não etiquetados. A aprendizagem mecânica sem supervisão pode encontrar padrões ou tendências que as pessoas não estão explicitamente à procura. Por exemplo, um programa de aprendizagem de máquina não supervisionado pode, através de dados de vendas *online*, procurar e identificar diferentes tipos de clientes que fazem compras [17].

A aprendizagem de máquina com **reforço** treina máquinas através da tentativa e erro para tomar a melhor ação, estabelecendo um sistema de recompensa. A aprendizagem com reforço pode treinar modelos para jogar jogos ou treinar veículos autónomos para conduzir, dizendo à máquina quando esta tomou as decisões certas, o que a ajuda a aprender ao longo do tempo que ações deve tomar [17].

Processamento de linguagem natural

O processamento da linguagem natural é um campo de aprendizagem de máquina em que as máquinas aprendem a compreender a linguagem natural falada e escrita pelos humanos, em vez dos dados e números normalmente utilizados para programar computadores. Isto permite às máquinas reconhecer a língua, compreender e responder, bem como criar novos textos e traduzir línguas [17].

A aprendizagem de máquina está também associada a várias tecnologias de inteligência artificial:

Redes neuronais

As redes neuronais são uma classe específica de algoritmos de aprendizagem de máquina, frequentemente utilizados. As redes neuronais artificiais são modeladas com base no cérebro humano, nas quais milhares ou milhões de nós de processamento estão interligados e organizados em camadas [17].

Numa rede neuronal artificial, as células, ou nós, estão ligadas, onde cada célula processa entradas e produz uma saída que é enviada para outros neurónios. Os dados etiquetados movem-se através dos nós, ou células, e cada célula desempenha uma função diferente. Numa rede neuronal treinada para identificar se uma imagem contém ou não um gato, os diferentes nós avaliam a informação e chegam a uma saída que indica se uma imagem apresenta um gato [17].

Aprendizagem profunda

As redes de aprendizagem profunda são redes neuronais com muitas camadas. Uma rede em camadas pode processar grandes quantidades de dados e determinar o “peso” de cada ligação na rede. Quanto mais camadas tiver, mais potencial tem para fazer coisas complexas corretamente [17]. Por exemplo, num sistema de reconhecimento de imagem, algumas camadas da rede neuronal podem detetar características individuais de um rosto, como olhos, nariz, ou boca, enquanto outra camada seria capaz de dizer se essas características aparecem de uma forma que indique um rosto [17].

Tal como as redes neuronais, a aprendizagem profunda é modelada na forma de como o cérebro humano funciona e alimenta muitas máquinas com aprendizagem, como veículos autónomos, e diagnósticos médicos [17].

A aprendizagem profunda requer um grande poder de computação, o que suscita preocupações quanto à sua sustentabilidade económica e ambiental [17].

3.2 U-Net

A Visão Computacional é um campo científico interdisciplinar que lida com a forma como os computadores podem ser programados para obter uma compreensão de alto nível a partir de imagens ou vídeos digitais. Do ponto de vista da engenharia, esta procura automatizar tarefas que o sistema visual humano pode fazer. A aprendizagem profunda permitiu que o campo da Visão Computacional avançasse rapidamente nos últimos anos [18]. Neste projeto, para identificar os pontos relevantes nas costas dos seres Humanos, foi utilizada uma arquitetura em particular, nomeadamente a U-Net, que utiliza um modelo de rede totalmente convolutivo.

U-Net é uma arquitetura desenvolvida por Olaf Ronneberger *et al*, para segmentação de imagens biomédicas, em 2015, na Universidade de Friburgo, Alemanha [19]. É uma das abordagens mais utilizadas atualmente em qualquer tarefa de segmentação

semântica. É uma rede neural totalmente convolutiva que foi concebida para aprender com menos dados de treino. Esta é uma melhoria em relação às *Fully Convolutional Networks* (FCN) existentes - “Fully convolutional networks for semantic segmentation” desenvolvida por Jonathan Long *et al*, em 2014 [20, 21].

3.2.1 Segmentação semântica

A segmentação semântica, também conhecida como uma classificação baseada em pixels, é uma tarefa importante na qual é classificado cada pixel de uma imagem como pertencente a uma determinada classe. Como é possível ver no exemplo da Figura 3.1, a segmentação semântica pode ser utilizada para extrair pessoas ou bicicletas a partir de imagens [22].

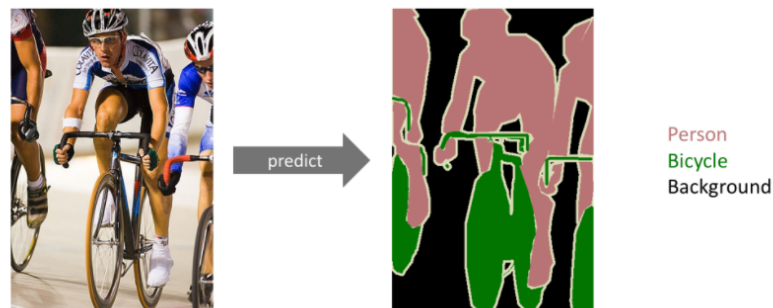


Figura 3.1: Exemplo de segmentação semântica [18]

O objetivo da segmentação semântica é o mesmo que a classificação tradicional da imagem na teledeteção, que é normalmente realizada através da aplicação de técnicas tradicionais de aprendizagem de máquina, tais como a *random forest* e o classificador *maximum likelihood*. Tal como na classificação de imagens, existem também duas entradas para a segmentação semântica [22]:

- uma imagem que contém várias camadas;
- uma imagem etiquetada que contém a classe para cada pixel.

3.2.2 U-Net - arquitetura

A U-Net é uma arquitetura de rede em forma de U, que consiste em quatro blocos codificadores e quatro blocos decodificadores que estão ligados através de uma ponte. A rede de codificadores divide em metade as dimensões espaciais e duplica o número de filtros (*feature channels*) a cada bloco. Da mesma forma, a rede decodificadora duplica as dimensões espaciais e divide em metade o número de *feature channels* [19, 21]. A Figura 3.2 mostra a arquitetura da rede U-Net.

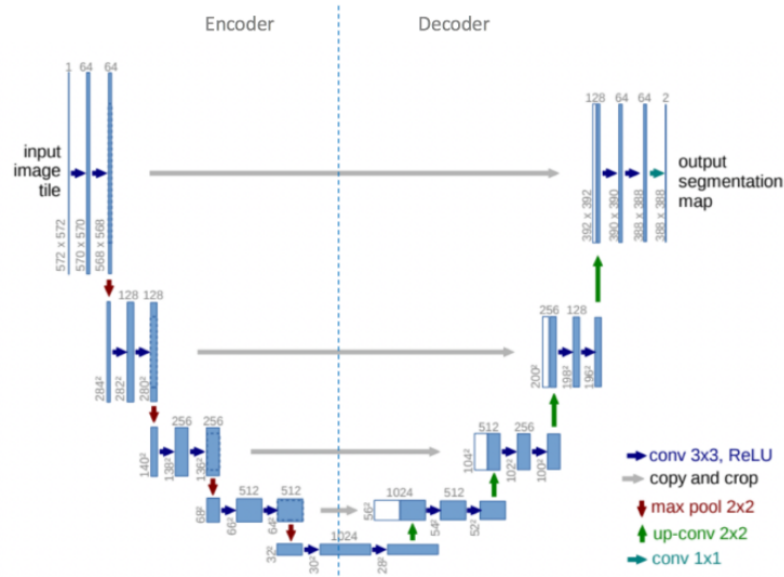


Figura 3.2: Arquitetura U-Net [19]

Rede codificadora

A rede codificadora é a primeira metade presente no diagrama da arquitetura da Figura 3.2. Esta é normalmente uma rede de classificação pré-formada como a MobileNet/InceptionResNet. Esta rede de codificadores funciona como extrator de características (*features*) e aprende uma representação abstrata da imagem de entrada através de uma sequência de blocos codificadores [19, 21].

Skip connection

Estas *skip connections* fornecem uma informação adicional que ajuda o decodificador a gerar melhores características semânticas. Estas atuam também como uma ligação de atalho que ajuda o fluxo indireto de gradientes para as camadas anteriores sem qualquer degradação. Em termos simples, é possível dizer que a *skip connection* ajuda a melhorar o fluxo de gradientes durante a retropropagação, o que por sua vez ajuda a rede a aprender uma melhor representação [19, 21].

Ponte

A ponte liga as redes codificadora e decodificadora e completa o fluxo de informação [19, 21].

Rede Descodificadora

A rede decodificadora é a segunda metade presente na arquitetura. Esta é utilizada para recolher a representação abstrata e gerar uma máscara da segmentação semântica [19, 21].

3.3 MobileNetV1

O modelo MobileNetV1 foi concebido para ser utilizado em aplicações móveis, e é o primeiro modelo de visão móvel por computador do TensorFlow [23, 24].

A MobileNetV1 utiliza convoluções profundas separadas. Esta reduz significativamente o número de parâmetros quando comparada com as redes com convoluções regulares com a mesma profundidade nas redes. Isto resulta em redes neuronais profundas e leves [23, 24].

A Figura 3.3 mostra que a rede MobileNet alcança, aproximadamente, a mesma percentagem de exatidão que as outras redes com um número menor de MAC que é a junção do número de operações de multiplicação e adição [23].

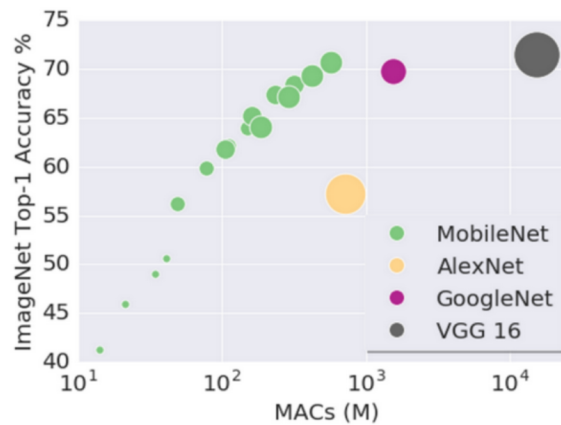


Figura 3.3: Exatidão da ImageNet Top-1 tendo em conta o número de *Multiply-Accumulates* (MAC) [23]

Uma convolução com profundidades separadas, descrita na Figura 3.4, é feita a partir de duas operações [23]:

- convolução com profundidade;
- convolução pontual.

No Anexo A estão disponibilizados mais detalhes sobre estas duas operações e sobre a arquitetura da MobileNetV1.

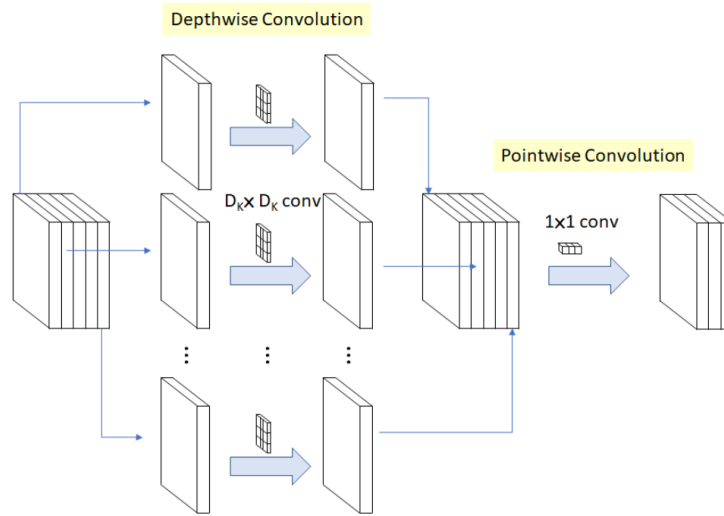


Figura 3.4: Convoluções profundas separadas [23]

A MobileNetV1 é uma classe de *Convolutional Neural Network* (CNN) que foi colocada como fonte aberta pelo Google e, por isso, proporciona um excelente ponto de partida para treinar classificadores que são extremamente pequenos e rápidos [23]. A Figura 3.5 mostra algumas das aplicações da MobileNet.



Figura 3.5: Aplicações da MobileNet [24]

3.3.1 MobileNetV2

A MobileNetV2 é uma melhoria significativa em relação à MobileNetV1 e impulsiona o desenvolvimento tecnológico para o reconhecimento visual móvel, incluindo classificação, detecção de objetos e segmentação semântica. A MobileNetV2 foi lançada como parte da biblioteca de classificação de imagens TensorFlow-Slim [25]. A Mobile-

NetV2 baseia-se nas ideias da MobileNetV1, utilizando a convolução profunda separada como blocos de construção eficientes. No entanto, esta introduz duas novas características na arquitetura [25]:

- resíduos invertidos;
- engarrafamentos lineares.

No Anexo B estão disponibilizados mais detalhes sobre estas duas novas características. A estrutura básica é mostrada abaixo, na Figura 3.6.

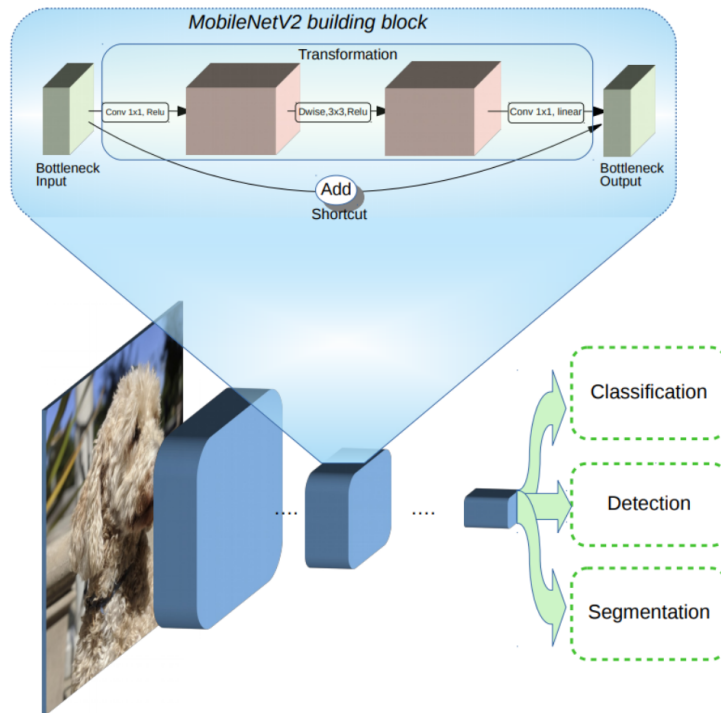


Figura 3.6: Visão geral da arquitetura da MobileNetV2 [25]

3.3.1.1 Arquitetura MobileNetV2

Na Figura 3.7 pode-se ver como os blocos de estrangulamento estão organizados. A variável t representa a taxa de expansão dos canais, onde utilizam um factor de 6, c representa o número de canais de entrada e n a frequência com que o bloco é repetido. Por fim, s mostra se a primeira repetição de um bloco utilizou um fator de 2 para o processo de *downsampling* [26].

Em suma, os modelos MobileNetV2 são mais rápidos para a mesma precisão em todo o espectro de latência. Em particular, os novos modelos utilizam duas vezes menos operações, necessitam de 30% menos parâmetros e são cerca de 30-40% mais rápidos

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	-

Figura 3.7: Arquitetura MobileNetV2 [26]

num telemóvel do que os modelos MobileNetV1, tudo isto ao mesmo tempo que atingem maior precisão [25], como é possível ver pela Figura 3.8.

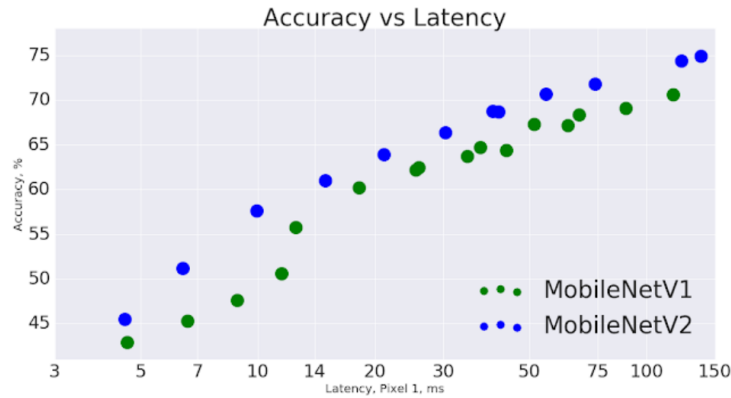


Figura 3.8: MobileNetV2 melhora a velocidade (latência reduzida) e aumenta a precisão da ImageNet Top 1 [25]

3.3.2 MobileNetV2-Unet

Tal como dito anteriormente, o modelo U-Net é composto por duas partes: uma rede codificadora e uma rede decodificadora. Este modelo funde características de alto nível e características de baixo nível através de uma *skip connection* entre as duas partes, o que melhora a capacidade de segmentar detalhes na imagem. Na MobileNetV2-UNet a rede codificadora é substituída pelas camadas de extração de características da MobileNetV2 pré-treinada. A rede decodificadora compreende quatro camadas de deconvolução e três blocos residuais invertidos, para que as dimensões de entrada e saída da MobileNetV2-UNet sejam idênticas. A arquitetura é mostrada na Figura 3.9 [27].

Algumas vantagens da MobileNetV2-UNet [28]:

- a MobileNetV2 tem menos parâmetros, pelo que é fácil de treinar;

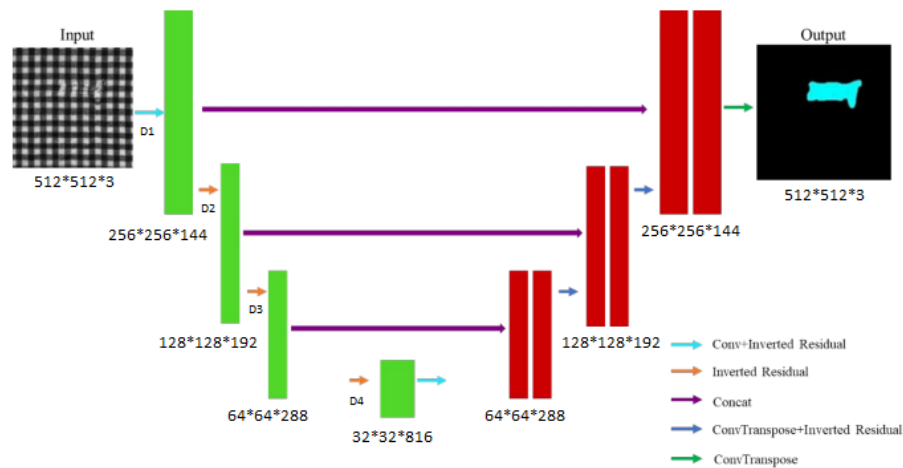


Figura 3.9: Arquitetura MobileNetV2-UNet [27]

- a utilização de um codificador pré-treinado ajuda o modelo a convergir muito mais rapidamente em comparação com o modelo não pré-treinado;
- um codificador pré-treinado ajuda o modelo a alcançar um desempenho elevado em comparação com um modelo não pré-treinado.

3.4 Inception

A rede Inception foi um passo importante no desenvolvimento dos classificadores CNN. Antes do seu aparecimento, a maioria das CNN populares empilhavam camadas de convolução cada vez mais profundas, na esperança de obter um melhor desempenho [29].

A rede Inception, por outro lado, é complexa (fortemente concebida). Esta utiliza muitas técnicas para aumentar o desempenho, tanto em termos de velocidade como de precisão. A sua constante evolução levou à criação de várias versões da rede. As versões mais populares são as seguintes [29]:

- Inception v1 [30];
- Inception v2 [31];
- Inception v3 [31];
- Inception v4 [32];
- Inception-ResNet [32].

Neste trabalho apenas vai ser utilizada a versão Inception-ResNet-v2, uma vez que esta é uma das versões mais recentes, e está disponível na interface Keras [33].

3.4.1 Inception-ResNet-v2

A Inception-ResNet-v2 é uma rede neuronal convolutiva que foi treinada com mais de um milhão de imagens da base de dados ImageNet. A rede tem 164 camadas de profundidade e pode classificar as imagens em 1000 diferentes categorias de objetos, tais como teclado, rato, lápis, e muitos animais. Como resultado, a rede aprendeu valiosas representações de características para uma vasta variedade de imagens [34].

Esta foi formulada com base numa combinação da estrutura Inception e da ligação Residual. No bloco Inception-ResNet, filtros convolucionais de múltiplos tamanhos são combinados com ligações residuais. A utilização de ligações residuais não só evita o problema de degradação causado por estruturas profundas, como também reduz o tempo de treino. A Figura 3.10 mostra a arquitetura básica de rede da Inception-ResNet-v2 [34, 32].

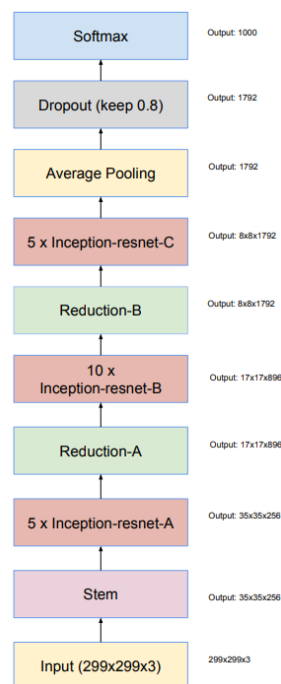


Figura 3.10: Arquitetura da rede Inception-ResNet-v2 [34]

Esta rede, comparada com a MobileNetV2, possui muitos mais parâmetros e uma maior profundidade, por outro lado, estas características resultam num tempo de inferência por etapa e tamanho maior. A Figura 3.11 apresenta um excerto da tabela presente no site do Keras com algumas das características mais importantes das redes neurais [33].

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0

Figura 3.11: Características de algumas redes neuronais presentes no Keras [33]

3.4.2 Inception-ResNet-V2-UNet

A Inception-ResNet-V2-UNet é uma combinação das arquiteturas Inception-ResNet-v2 e U-Net. Tal como na MobileNetV2-UNet, a parte da rede codificadora é substituída pelas camadas de extração de características da Inception-ResNet-V2 pré-treinada. Esta rede também partilha algumas das vantagens da MobileNetV2-UNet, tais como, a utilização de um codificador pré-treinado ajudar o modelo a convergir muito mais rapidamente e a ter um desempenho mais elevado.

3.5 Aumento dos dados

O aumento dos dados é um conjunto de técnicas utilizadas para aumentar artificialmente a quantidade de dados, através da geração de novos dados a partir dos dados existentes. Isto inclui fazer pequenas alterações aos dados ou utilizar modelos de aprendizagem profunda para gerar novos dados [35].

As aplicações de aprendizagem de máquina, especialmente no domínio da aprendizagem profunda, continuam a diversificar-se e a aumentar rapidamente. As técnicas de

aumento dos dados podem ser uma boa ferramenta contra os desafios que o mundo da inteligência artificial enfrenta [35].

O aumento dos dados é útil para melhorar o desempenho e os resultados dos modelos de aprendizagem de máquina, formando novos e diferentes exemplos para treinar conjuntos de dados. Se o conjunto de dados de um modelo de aprendizagem de máquina for rico e suficiente, o modelo tem um desempenho melhor e mais preciso [35].

Para modelos de aprendizagem de máquina, a recolha e etiquetagem de dados pode ser um processo cansativo e dispendioso. As transformações nos conjuntos de dados, utilizando técnicas de aumento dos dados, permitem às empresas reduzir estes custos operacionais [35].

Uma das etapas de um modelo de dados é a limpeza de dados que é necessária para modelos de alta precisão. Contudo, se a limpeza reduzir a representatividade dos dados, então o modelo não pode fornecer boas previsões para as entradas do mundo real. As técnicas de aumento dos dados permitem que os modelos de aprendizagem de máquina sejam mais robustos, criando variações que o modelo pode ver no mundo real [35]. A Figura 3.12 mostra como as técnicas de aumento de dados são introduzidas no *Dataset* antes de este ser treinado.

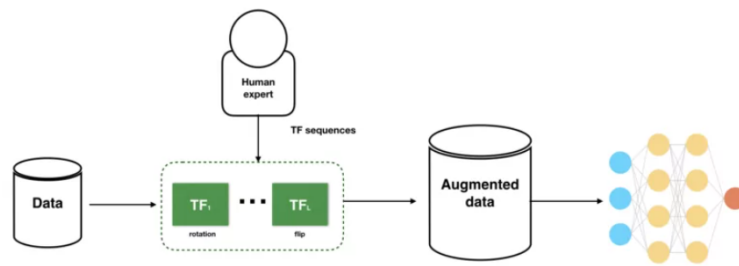


Figura 3.12: Aumento de dados [35]

Para aumentar os dados, o mais popular a fazer são simples alterações nos dados visuais. As técnicas clássicas de processamento de imagem para o aumento dos dados são [35]:

- preenchimento;
- rotação aleatória;
- redimensionamento;
- inverter verticalmente e horizontalmente;
- translação (a imagem é movida na direcção X, Y);
- ampliação;

- escurecimento e modificação do brilho/cor;
- converter para tons de cinza;
- variar contraste;
- adicionar ruído;
- apagamento aleatório;
- recortar.

Os benefícios do aumento dos dados incluem [35]:

- melhora a precisão da previsão do modelo:
 - adiciona mais dados de treino aos modelos;
 - previne a escassez de dados para melhores modelos;
 - reduz o *overfitting* de dados e cria variabilidade nos dados;
 - aumenta a capacidade de generalização dos modelos;
 - ajuda a resolver questões de desequilíbrio de classe na classificação;
- reduz os custos de recolha e etiquetagem de dados;
- permite a previsão de eventos raros;
- previne problemas de privacidade de dados.

3.6 Otimizador Adam

Um otimizador é um dos dois argumentos necessários para compilar um modelo Keras. O *Adaptive Moment Estimation* (Adam) é um algoritmo para a otimização da técnica da descida por gradiente. Este método é realmente eficiente quando se trabalha com problemas que envolvem muitos dados ou parâmetros, devido a utilizar menos memória. Intuitivamente, é uma combinação do algoritmo de descida por gradiente com momento e o algoritmo *Root Mean Square Propagation* (RMSP) [36] [37]. O Código 3.1 mostra como é possível utilizar este otimizador com o Keras.

Código 3.1: Otimizador Adam [38]

```
1 | tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9,  
  | beta_2=0.999, epsilon=1e-07, amsgrad=False, name="Adam", **  
  | kwargs)
```

A descida por gradiente é um algoritmo de otimização iterativo, utilizado para encontrar o valor mínimo para uma função. A ideia geral é inicializar os parâmetros para valores aleatórios, e depois dar pequenos passos na direção do declive em cada iteração. A descida por gradiente é altamente utilizada na aprendizagem supervisionada para minimizar a função de erro e encontrar os valores ideais para os parâmetros. Várias extensões foram concebidas para os algoritmos de descida por gradiente. O otimizador Adam envolve uma combinação de duas dessas metodologias [39, 36]:

Momentum

Este algoritmo é utilizado para acelerar o algoritmo de descida por gradiente tendo em consideração a “média exponencialmente ponderada” dos gradientes. A utilização de médias faz o algoritmo convergir para os mínimos a um ritmo mais rápido [36]. A equação 3.1 mostra como se calculam os “pesos” no instante $t + 1$ através do algoritmo *momentum*. Esta equação engloba a equação 3.2 para calcular o agregado de gradientes no instante atual.

$$w_{t+1} = w_t - \alpha_t m_t \quad (3.1)$$

onde,

$$m_t = \beta m_{t-1} + (1 - \beta) \left[\frac{\delta L}{\delta w_t} \right] \quad (3.2)$$

m_t - agregado de gradientes no instante t (atual), inicialmente, $m_t = 0$

m_{t-1} - agregado de gradientes no instante $t - 1$ (anterior)

w_t - pesos no instante t

w_{t+1} - pesos no instante $t + 1$

α_t - taxa de aprendizagem no instante t

δL - derivação da Função de Perda

δw_t - derivação dos pesos no instante t

β - parâmetro de média móvel (constante, 0,9)

Root Mean Square Propagation

RMSP ou RMSprop é um algoritmo de aprendizagem adaptativo que tenta melhorar o *Adaptive Gradient Algorithm* (AdaGrad). Em vez de utilizar a soma acumulada de gradientes ao quadrado, como no AdaGrad, utiliza a média móvel exponencial [36]. A equação 3.3 mostra como se calculam os “pesos” no instante $t + 1$ através do algoritmo RMSP. Esta equação engloba a equação 3.4 para calcular a soma do quadrado dos gradientes passados.

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t + \epsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t} \right] \quad (3.3)$$

onde,

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[\frac{\delta L}{\delta w_t} \right]^2 \quad (3.4)$$

v_t - soma do quadrado dos gradientes passados, (i.e. $\sum(\frac{\delta L}{\delta w_{t-1}})$), inicialmente, $v_t = 0$)

ϵ - uma pequena constante positiva (10^{-8})

O otimizador Adam herda os pontos fortes dos dois métodos acima referidos e baseia-se neles para proporcionar uma descida de gradiente mais otimizada [36, 37].

Como é possível ver pela Figura 3.13, a taxa de descida por gradiente é controlada de modo a que haja uma oscilação mínima quando esta atinge o mínimo global, onde ao mesmo tempo são dados passos suficientemente grandes (tamanho do degrau) de modo a passar os mínimos obstáculos locais ao longo do caminho. Assim, combinando as características dos métodos acima referidos é possível atingir o mínimo global de forma eficiente [36, 37].

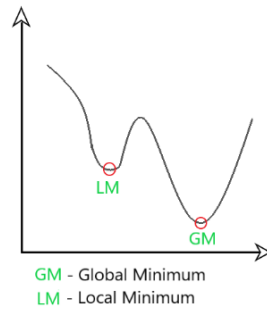


Figura 3.13: Otimizador Adam [36]

Considerando as expressões utilizadas nos dois métodos acima referidos, são obtidas as equações 3.5 e 3.6 [36, 37]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) * \left[\frac{\delta L}{\delta w_t} \right]^2 \quad (3.5)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) * \left[\frac{\delta L}{\delta w_t} \right]^2 \quad (3.6)$$

Parâmetros usados:

ϵ - Pequena constante positiva para evitar uma divisão por 0 quando $v_t = 0$ (10^{-8})

β_1 e β_2 - taxas de diminuição da média dos gradientes nos dois métodos acima referidos ($\beta_1 = 0,9$ e $\beta_2 = 0,999$)

α - taxa de aprendizagem (0,001)

Uma vez que tanto o m_t como o v_t são inicializados como 0, com base nos métodos acima referidos, observa-se que ambos ganham uma tendência para 0, da mesma forma que β_1 e β_2 são aproximadamente 1. Este otimizador corrige este problema através da computação de m_t e v_t (corrigidos por enviesamento), demonstrados nas equações 3.7 e 3.8, respetivamente. Isto também é realizado para controlar os pesos enquanto se atinge o mínimo global para evitar oscilações elevadas quando perto deste. As equações utilizadas são as seguintes [36, 37]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (3.7)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (3.8)$$

Parâmetros usados:

$\hat{m}_t$ - agregado de gradientes no instante atual corrigido por enviesamento

$\hat{v}_t$ - soma do quadrado dos gradientes passados corrigido por enviesamento.

Por intuição, o algoritmo adapta-se à descida por gradiente após cada iteração, para que esta permaneça controlada e imparcial durante todo o processo, daí o nome Adam [36, 37].

De seguida, em vez de utilizar os parâmetros de peso normais m_t e v_t , são assumidos os parâmetros de peso corrigidos por enviesamento $\hat{m}_t$ e $\hat{v}_t$, obtendo a equação 3.9 [36, 37]:

$$w_{t+1} = w_t - \hat{m}_t \left(\frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \right) \quad (3.9)$$

Com base nos pontos fortes dos modelos anteriores, o otimizador Adam proporciona um desempenho muito mais elevado e supera os outros algoritmos por uma grande margem para dar uma descida de gradiente otimizada. A Figura 3.14 mostra claramente como o otimizador Adam supera o resto dos otimizadores por uma margem considerável em termos de custo de treino (baixo) e desempenho (alto) [36, 37].

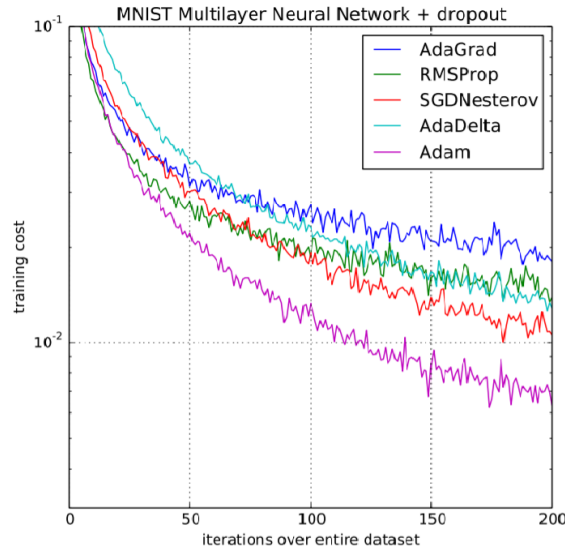


Figura 3.14: Comparação do desempenho no custo do treino [36]

3.7 Função de perda

As funções de perda são funções que mostram o desempenho do modelo e são utilizadas para treinar um modelo de aprendizagem máquina (utilizando um tipo de otimização), e são normalmente diferenciáveis nos parâmetros do modelo.

3.7.1 Dice loss

Dice loss tem origem no *Sørensen–Dice coefficient* (DSC), que é uma estatística desenvolvida nos anos 40 para medir a semelhança entre duas amostras. Este foi apresentado à comunidade de visão computacional por Milletari *et al.* em 2016 para a segmentação de imagens médicas em 3D [40].

A equação 3.10 mostra a definição do DSC, em que p_i e g_i representam pares dos valores de previsão dos pixels e dos pixels verdadeiros, respetivamente. No cenário de detecção de limites, os valores de p_i e g_i são 1 ou 0, representando se o pixel é um limite (valor de 1) ou não (valor de 0). Portanto, o denominador é a soma do total de pixels de limite tanto da previsão como da verdade, e o numerador é a soma dos pixels de limite corretamente previstos porque a soma só aumenta quando p_i e g_i coincidem (ambos de valor 1) [40].

$$D = \frac{2 \sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2} \quad (3.10)$$

A Figura 3.15 é outra visão da equação 3.10 da perspectiva da teoria de conjuntos, na qual o DSC é uma medida da sobreposição entre dois conjuntos. Por exemplo, se

dois conjuntos A e B se sobrepõem perfeitamente, o DSC obtém o seu valor máximo, 1. Caso contrário, o DSC começa a diminuir, chegando ao seu valor mínimo, 0, se os dois conjuntos não se sobrepuserem de todo. Portanto, o intervalo do DSC está entre 0 e 1 [40].

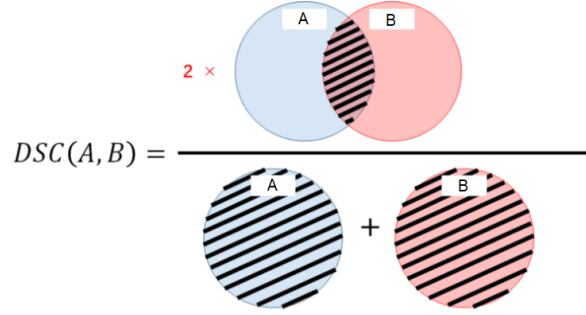


Figura 3.15: *Sørensen–Dice coefficient*

]Sørensen–Dice coefficient [40]

Assim, a *Dice loss* considera a informação da perda tanto local como globalmente, o que é crítico para obter uma elevada precisão [40].

3.8 Métricas de classificação

A escolha das métricas de classificação certas é crucial para avaliar os modelos de aprendizagem de máquina. Em algumas aplicações, a análise de uma única métrica pode não dar o panorama completo do problema a resolver, e pode ser necessário utilizar um conjunto de métricas para ter uma avaliação concreta dos seus modelos.

Estas métricas são utilizadas para monitorizar e medir o desempenho de um modelo durante o treino e o teste.

3.8.1 Jaccard

Dados os valores previstos como ($\hat{y}$) e os valores reais como y , o índice *Jaccard* pode ser definido como [41]:

$$j(y, \hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} = \frac{|y \cap \hat{y}|}{|y| + |\hat{y}| - |y \cap \hat{y}|} \quad (3.11)$$

Por isso, com o seguinte conjunto de valores previstos e reais.

$$previstos = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]$$

$$reais = [1, 1, 0, 0, 0, 1, 1, 1, 1, 1]$$

O índice *Jaccard* será:

$$j(y, \hat{y}) = \frac{8}{10 + 10 - 8} = 0,66 \quad (3.12)$$

A ideia por detrás deste índice é que quanto maior for a semelhança entre estes dois grupos, maior este índice será [41].

3.8.2 Intersection over union

A métrica *Intersection over Union* (IoU), também conhecida como índice *Jaccard*, é essencialmente um método utilizado para calcular a sobreposição percentual entre a máscara objetivo e a previsão dos resultados. Esta métrica está também intimamente relacionada com o DSC, que foi visto anteriormente [42].

Simplesmente, a métrica IoU mede o número de pixels comuns entre a máscara objetivo e a máscara prevista dividido pelo número total de pixels presentes em ambas as máscaras [42].

Por exemplo, para calcular a pontuação IoU da seguinte previsão, dada a máscara objetivo etiquetada, na Figura 3.16.

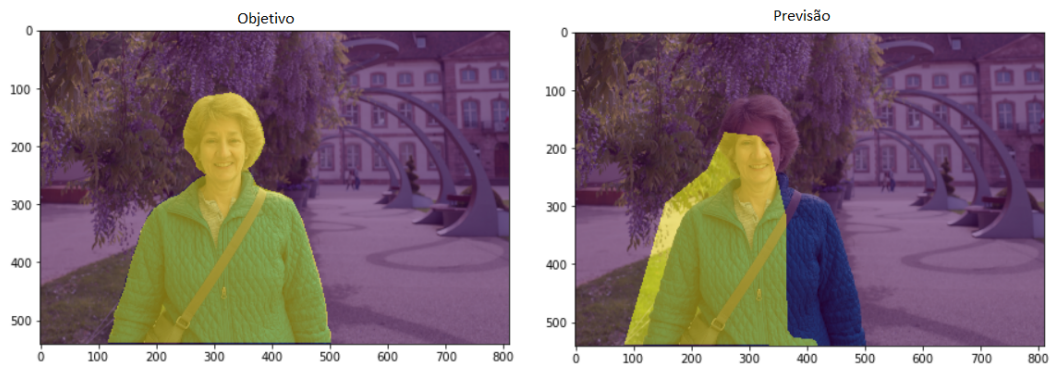


Figura 3.16: Máscara objetivo e prevista [42]

A intersecção ($A \cap B$) é composta pelos pixels encontrados tanto na máscara prevista como na máscara objetivo, enquanto a união ($A \cup B$) é composta por todos os pixels encontrados tanto na previsão como na máscara objetivo, como é possível ver na Figura 3.17 [42].

3.8.3 Precision

Precision é a relação entre os positivos verdadeiros e todos os positivos [43]. Matematicamente:

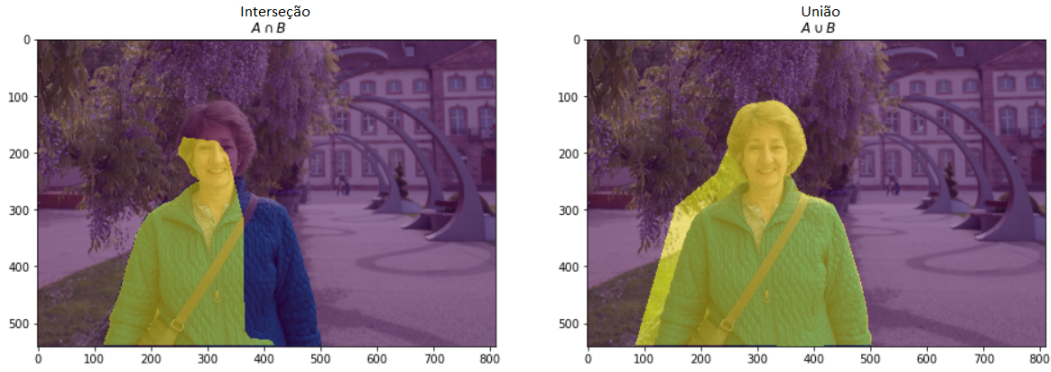


Figura 3.17: Interseção e união das máscaras [42]

$$Precision = \frac{Positivos\ Verdadeiros\ (PV)}{Positivos\ Verdadeiros\ (PV) + Positivos\ Falsos\ (PF)} \quad (3.13)$$

A *precision* também dá uma medida dos pontos de informação relevantes [43].

3.8.4 Recall

Recall é a medida de como o modelo identifica correctamente os positivos verdadeiros. Neste caso, para os pontos revelantes presentes nas costas, a *Recall* diz quantos desses pontos foram identificados corretamente [43]. Matematicamente:

$$Recall = \frac{Positivos\ Verdadeiros\ (PV)}{Positivos\ Verdadeiros\ (PV) + Negativos\ Falsos\ (NF)} \quad (3.14)$$

3.8.5 Accuracy

Accuracy é a razão entre o número total de previsões corretas e o número total de previsões [43].

$$Accuracy = \frac{PV + Negativos\ Verdadeiros\ (NV)}{PV + PF + NV + NF} \quad (3.15)$$

3.8.6 F1-Score

F1-Score é a média harmónica entre a *precision* e o *Recall* [43].

$$F1 = 2 \frac{Precision * Recall}{Precision + Recall} \quad (3.16)$$

Uma boa pontuação de *F1-Score* é indicativo de uma boa *precision* e de um bom valor de *Recall* também.

3.9 Conclusão do capítulo 3

Neste capítulo foram apresentadas uma pequena introdução ao que é a aprendizagem de máquina e a aprendizagem profunda, uma vez que este projeto se enquadra nessa área, os modelos de previsão utilizados, e as ferramentas utilizadas para treinar estes modelos.

Capítulo 4

Hardware e Interfaces

Neste capítulo são apresentados os vários dispositivos e interfaces utilizados para a solução da plataforma para auscultação autónoma. O dispositivo principal é o computador, responsável por treinar e avaliar o modelo de previsão dos pontos revelantes nas costas das pessoas, calcular as coordenadas espaciais destes pontos, calcular o offset do alinhamento entre a câmara OAK-D e o braço robótico, receber as imagens desta mesma câmara, e enviar as instruções para o braço robótico UR3e, um robô colaborativo da Universal Robots. As interfaces utilizadas foram o URsim para simular e testar o movimento do braço robótico através de alguns códigos desenvolvidos e instruções diretamente colocadas na interface, o TensorFlow e o Keras para avaliar, treinar e utilizar o modelo de previsão, a DepthAI para comunicar com a câmara, e os sockets e o RTDE para comunicar com o braço robótico.

4.1 Hardware

Nesta secção vão ser apresentados os dispositivos utilizados na construção deste projeto. Na Figura 4.1 é possível ver a arquitetura geral.

4.1.1 Manipulador robótico UR3e

O dispositivo robótico utilizado foi o manipulador industrial UR3e, presente na Figura 4.2, produzido pela Universal Robots. Apesar de ter um curto alcance, optou-se por este manipulador por ser o único disponível.

O robô colaborativo UR3e de mesa é útil para tarefas mais leves como a montagem ou para cenários de bancada em trabalhos automatizados. Este pesa apenas 11 kg, mas tem uma carga útil de 3 kg, 360° de rotação em todas as articulações do braço, e uma rotação infinita na articulação final [44]. Estas características e outras presentes na Tabela 4.1

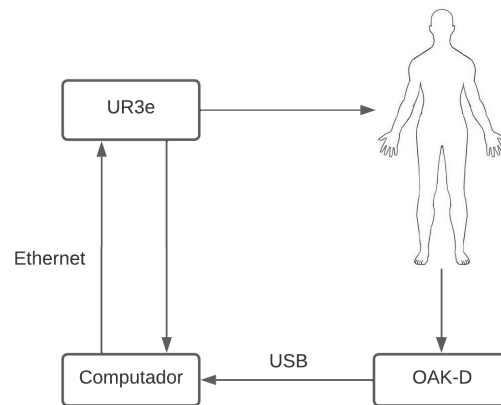


Figura 4.1: Arquitetura geral



Figura 4.2: Robô UR3e [44]

tornam o *cobot* UR3e um robô de mesa flexível e leve para trabalhar lado a lado com os empregados no mercado atual [44].

O robô UR3e pode ser utilizado em montagens, polimento, aplicação de cola, e aplicações de parafusos que requerem uma qualidade de produto uniforme. Este também pode ser utilizado numa estação de trabalho separada, recolhendo, montando e colocando peças em fluxos de produção otimizados. Devido à sua forma compacta e de fácil programação, é fácil alternar entre tarefas para satisfazer as necessidades de fabrico [44].

As aplicações do robô UR3e abrangem indústrias de fabrico desde dispositivos médicos, a placas de circuitos, e componentes eletrónicos [44].

4.1.2 Computador

Utilizou-se um computador pessoal para comunicar com o controlador do UR3e, com as especificações apresentadas na Tabela 4.2.

A comunicação entre o computador e o robô foi estabelecida através de um cabo de rede Ethernet. Este permite interconectar dispositivos para que ocorra troca de informa-

Tabela 4.1: Especificações principais do manipulador robótico UR3e [45]

Graus de liberdade	6
Carga útil	3 kg / 6,6 lbs
Repetibilidade	$\pm 0,03$, com carga útil, por ISO 9283
Peso com o cabo	11,2 kg / 24,7 lbs
Alcance	500 mm / 19,7 in
Alcance das articulações	Base: $\pm 360^\circ$ Ombro: $\pm 360^\circ$ Cotovelo: $\pm 360^\circ$ Pulso 1: $\pm 360^\circ$ Pulso 2: $\pm 360^\circ$ Pulso 3: Infinito
Velocidade máxima	Base: $\pm 180^\circ/\text{Seg.}$ Ombro: $\pm 180^\circ/\text{Seg.}$ Cotovelo: $\pm 180^\circ/\text{Seg.}$ Pulso 1: $\pm 360^\circ/\text{Seg.}$ Pulso 2: $\pm 360^\circ/\text{Seg.}$ Pulso 3: $\pm 360^\circ/\text{Seg.}$
Fonte de alimentação	100-240 VAC, 47-40 Hz
Temperatura de funcionamento	0-50°
Classificação IP	IP54
Programação	Polyscope
Sensor F/T - Força, $x - y - z$	
Alcance	10 Nm
Resolução	0,02 Nm
Precisão	0,10 Nm

Tabela 4.2: Especificações do computador utilizado

Modelo	Lenovo Legion Y530-15ich
Processador	Intel Core i5-8300H 2,30 GHz
Memória RAM	8 Gb
Disco Rígido	500 Gb SSD
Placa Gráfica	NVIDIA GeForce GTX 1050 4 Gb
Sistema Operativo	Windows 10, 64 bit

ção entre os mesmos. A comunicação entre o computador e a câmera foi estabelecida através de um cabo *Universal Serial Bus* (USB) *Type C*.

4.1.3 Câmera OAK-D

A placa de base OAK-D tem três câmeras a bordo que implementam visão estéreo e *Red-Green-Blue* (RGB), canalizadas diretamente para o *System on Module* (SoM) do *OpenCV AI Kit* (OAK) para processamento de profundidade e Inteligência Artificial

(IA). Esta câmera utiliza o cabo USB-C para enviar dados para um *host* e alimentação. Suporta tanto USB2 como USB3 (5 Gbps / 10 Gbps) [46]. Na Tabela 4.3 estão algumas especificações desta câmera.

Tabela 4.3: Especificações da câmera OAK-D [46]

Especificações	Câmera colorida	Par estéreo
Sensor	IMX378	OV9282
DFOV / HFOV / VFOV	81° / 69° / 55°	82° / 72° / 50°
Resolução	12 MP (4032x3040), 4 k	1 MP (1280x800)
Foco	Auto-Foco: 8 cm - ∞	Foco-Fixo: 19,6 cm - ∞
Máximo <i>framerate</i>	60 FPS	120 FPS
Número-F	2,0	2,2
Tamanho da lente	1/2,3 inch	1/4 inch
Tamanho do pixel	1,55 μm x 1,55 μm	3 μm x 3 μm

A câmera OAK-D possui uma Unidade de Processamento Visual (UPV) Intel Myriad X. Este é um processador capaz de executar redes neuronais modernas para percepção visual, e simultaneamente criar um mapa de profundidade a partir do par de imagens estéreo em tempo real [46, 47].

Esta câmera está a ser utilizada para desenvolver aplicações numa grande variedade de áreas. É possível ver aplicações nas áreas da educação, cuidados neonatais, tecnologias de assistência para deficientes, *Augmented Reality/Virtual Reality* (AR/VR), inspeção de armazéns utilizando drones, robótica, agricultura, análise desportiva, retalho, e até mesmo publicidade [46, 47].

4.2 Interfaces

Todo o código desenvolvido neste projeto foi escrito em Python 3.7 com a ajuda do *software* Visual Studio Code. Para programar o manipulador robótico UR3e é possível fazê-lo através do *teach pedant*, do envio de instruções URScript a partir de um computador e através de uma *Application Programming Interface* (API) em C. Tendo em conta os objetivos deste projeto, e da forma como o manipulador robótico está inserido na arquitetura de *hardware*, escolheu-se programar o UR3e através de URScripts. O computador foi conectado ao robô a partir de uma aplicação cliente no computador, ligando o controlador do robô via *sockets Transmission Control Protocol/Internet Protocol* (TCP/IP) e através da interface *Real-Time Data Exchange* (RTDE). A conexão foi feita, como já mencionado acima, com um cabo de rede, usando as portas 30002 para a comunicação com *sockets* e 30004 para a interface RTDE. A frequência de comunicação na porta 30002 era 10 Hz e na porta 30004 era de 125 Hz [48, 49].

Utilizou-se também o *teach pedant*, por ser mais prático e direto, para testar e obter coordenadas ou ângulos do braço robótico, movimentar o robô para posições desejadas,

e alterar o modo de operação do robô.

Abaixo vão ser descritas todas as interfaces utilizadas neste projeto.

4.2.1 URsim

Para testar alguns códigos desenvolvidos antes de os enviar para o manipulador robótico físico, ou seja, planejar trajetórias ou testar coordenadas, foi utilizado o *software* URSim.

O URSim é um *software* de simulação que é utilizado para programação *offline* e simulação de programas de robôs [50].

A grande vantagem deste *software* é o facto de ser possível programar o braço robótico sem ser necessário o programador se movimentar para perto do mesmo. Mas, apesar disso, este simulador tem algumas limitações, uma vez que nenhum braço robótico real está ligado, especialmente no controlo de força que será limitado na sua utilização. Abaixo estão descritas algumas funções que não funcionam em comparação com um robô real [50].

- a paragem de emergência não pode ser utilizada;
- as entradas analógicas não podem ser definidas;
- as trajetórias são perfeitas;
- as colisões com o próprio ou com objetos circundantes não funcionam;
- o modo de força não funciona.

Este simulador funciona de igual forma como o *teach pendant*. Para ligar ao robô deste *software* basta conectar através de *sockets*, utilizando o IP que é fornecido do robô simulado. Uma vez que o URsim foi feito para o sistema operativo Linux, e o projeto foi realizado no sistema operativo Windows, foi necessário utilizar uma máquina virtual para instalar o sistema operativo Linux [50]. A Figura 4.3 mostra uma das várias funcionalidades do simulador.

4.2.2 TensorFlow

Criado pela equipa da Google Brain, o TensorFlow é uma plataforma de código aberto para computação numérica e aprendizagem de máquina em grande escala. O TensorFlow reúne uma série de modelos e algoritmos de aprendizagem de máquina e de aprendizagem profunda e torna-os úteis por meio de metáforas programáticas comuns. Este utiliza o Python ou o JavaScript para fornecer uma API conveniente para construir aplicações, enquanto executa essas aplicações em C++ de alto desempenho [51].

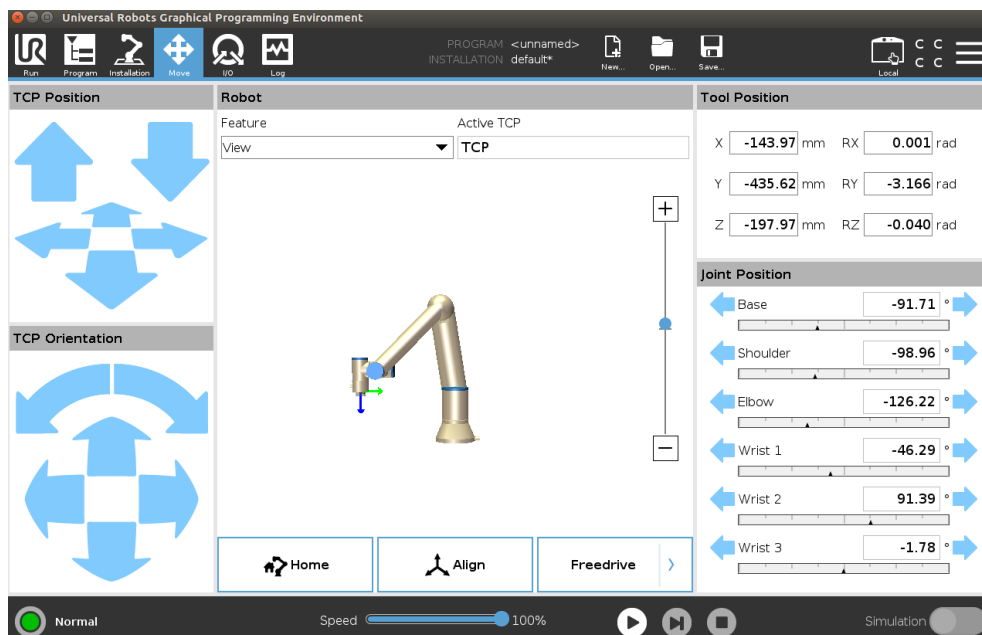


Figura 4.3: Simulador URSim

Esta plataforma compete com outras plataformas, tais como, PyTorch [52] e Apache MXNet [53]. Nesta é possível treinar e executar redes neurais para classificação de dígitos manuscritos, reconhecimento de imagem, incorporação de palavras, redes neurais recorrentes, modelos sequenciais para tradução automática, e processamento de linguagem natural. Ainda mais, o TensorFlow suporta a previsão da produção à escala, com os mesmos modelos utilizados para o treino, e tem também uma vasta biblioteca de modelos pré-treinados que podem ser utilizados em qualquer tipo de projetos [51].

O TensorFlow permite aos programadores criar gráficos de fluxo de dados que descrevem como os dados se movem através de um grafo, ou uma série de nós de processamento. Cada nó no gráfico representa uma operação matemática, e cada ligação ou margem entre nós é uma matriz de dados multidimensional, ou tensor [51].

As aplicações TensorFlow podem ser executadas em vários dispositivos: uma máquina local, um *cluster* na nuvem, dispositivos iOS e Android, *Central Processing Unit* (CPU) ou *Graphics Processing Unit* (GPU). Utilizando a própria nuvem da Google, o TensorFlow pode ser executado na *Tensor Processing Unit* (TPU) de silício personalizada da Google, para uma maior aceleração. Os modelos resultantes criados pelo TensorFlow, no entanto, podem ser implantados na maioria dos dispositivos onde serão utilizados para fazer previsões [51].

O TensorFlow 2.0, lançado em Outubro de 2019, renovou de muitas maneiras a sua estrutura com base na opinião dos utilizadores, para facilitar o seu trabalho (como, por exemplo, utilizando a API Keras para o treino do modelo) e obter mais desempenho. A distribuição da treinos é mais fácil de executar graças a uma nova API, e o suporte para

o TensorFlow Lite torna possível a implementação de modelos numa maior variedade de plataformas. Contudo, o código escrito nas versões anteriores do TensorFlow deve ser reescrito, para tirar o máximo proveito das novas características do TensorFlow 2.0 [51].

Em suma, o TensorFlow pode ser interpretado como uma camada de infra-estrutura para uma programação diferenciável, combinando quatro capacidades essenciais [54]:

- executa eficientemente operações de tensor de baixo nível na CPU, GPU, ou TPU;
- computação do gradiente de expressões diferenciáveis arbitrárias;
- computação em escala para muitos dispositivos, tais como *clusters* de centenas de GPU;
- exportação de programas (“gráficos”) para tempos de execução externos, tais como servidores, navegadores, dispositivos móveis e embutidos.

4.2.2.1 Keras

O Keras é uma API de aprendizagem profunda de alto nível escrita em Python, que funciona em cima da plataforma de aprendizagem de máquina TensorFlow. Este é uma interface acessível e produtiva para a resolução de problemas de aprendizagem de máquina, com foco na aprendizagem profunda moderna. Fornece abstrações essenciais e blocos de construção para o desenvolvimento de soluções de aprendizagem de máquina com alta velocidade de iteração. Foi também desenvolvido com o objetivo de permitir uma experimentação rápida, sendo capaz de ir da ideia ao resultado o mais rápido possível, proporcionando uma ferramenta fundamental para fazer uma boa investigação [54].

O Keras possui as seguintes características [54]:

- simples - mas não simplista. O Keras reduz a carga cognitiva do desenvolvedor, para libertar o programador de forma a este se concentrar nas partes do problema que realmente importam;
- flexível - adota o princípio da divulgação progressiva da complexidade: os fluxos simples de trabalho devem ser rápidos e fáceis, enquanto os fluxos arbitrariamente avançados de trabalho devem ser possíveis através de um caminho claro que se baseia no que já se aprendeu;
- poderoso - proporciona um grande desempenho e escalabilidade na indústria: é utilizado por organizações e empresas incluindo a National Aeronautics and Space Administration (NASA), YouTube, ou Waymo.

O Keras permite aos engenheiros e investigadores tirar o máximo partido da escalabilidade e das capacidades de *cross-platform* do TensorFlow, sendo possível executar o Keras em TPU ou em grandes grupos de GPU, e exportar os modelos Keras para executar no *browser* ou num dispositivo móvel [54].

As estruturas de dados centrais do Keras são camadas e modelos. O tipo de modelo mais simples é o Sequencial, que representa uma sequência linear de camadas [54].

O Código 4.1 mostra como utilizar um modelo Sequencial.

Código 4.1: Modelo Sequencial [54]

```
1 from tensorflow.keras.models import Sequential
2
3 model = Sequential()
```

A tarefa de acumulação de camadas é possível através da função *add()*, como mostra o Código 4.2:

Código 4.2: Acumulação de camadas [54]

```
1 from tensorflow.keras.layers import Dense
2
3 model.add(Dense(units=64, activation='relu'))
4 model.add(Dense(units=10, activation='softmax'))
```

Depois do modelo estar completo, é possível configurar o seu processo de aprendizagem com a função *compile()*, como mostra o Código 4.3. A filosofia do Keras é manter as coisas simples, permitindo ao mesmo tempo que o programador tenha o controlo total quando necessário [54].

Código 4.3: Configuração do Modelo [54]

```
1 model.compile(loss=dice_loss, optimizer=Adam(lr), metrics=[
    dice_coef, iou, Recall(), Precision()])
```

Depois de ter o modelo configurado, pode-se iterar os seus dados de treino em lotes com a função *fit()*, como mostra o Código 4.4:

Código 4.4: Treino do modelo [54]

```
1 model.fit(train_dataset, epochs=num_epochs, validation_data=
    valid_dataset, callbacks=callbacks)
```

Através do Código 4.5, pode-se gerar previsões sobre novos dados ou no *Dataset* de teste, e avaliar esta mesma previsão através das métricas presentes no Código 4.6

Código 4.5: Previsões sobre novos dados [54]

```
1 y_pred = model.predict(np.expand_dims(imagem_def, axis=0))[0]
```

Código 4.6: Avaliação do modelo no *Dataset* de teste [54]

```

1 | acc_value = accuracy_score(y, y_pred)
2 | f1_value = f1_score(y, y_pred, labels=np.unique(y_pred),
   |             average="binary")
3 | jac_value = jaccard_score(y, y_pred, labels=np.unique(y_pred),
   |             average="binary")
4 | recall_value = recall_score(y, y_pred, labels=np.unique(y_pred),
   |             average="binary")
5 | precision_value = precision_score(y, y_pred, labels=np.unique(
   |             y_pred), average="binary")

```

O que está acima descrito é a forma mais elementar de usar o Keras. No entanto, este também é um sistema flexível, adequado para iterar sobre ideias de investigação de ponta. O Keras segue o princípio da divulgação progressiva da complexidade: facilita o início, mas possibilita o tratamento de casos de utilização arbitrariamente avançados, exigindo apenas uma aprendizagem incremental em cada passo [54].

4.2.3 DepthAI

A API DepthAI permite aos programadores ligarem-se, configurarem e comunicarem com os seus dispositivos OAK, suportando tanto as API Python como C++ [55]. A Figura 4.4 mostra a arquitetura da API DepthAI.

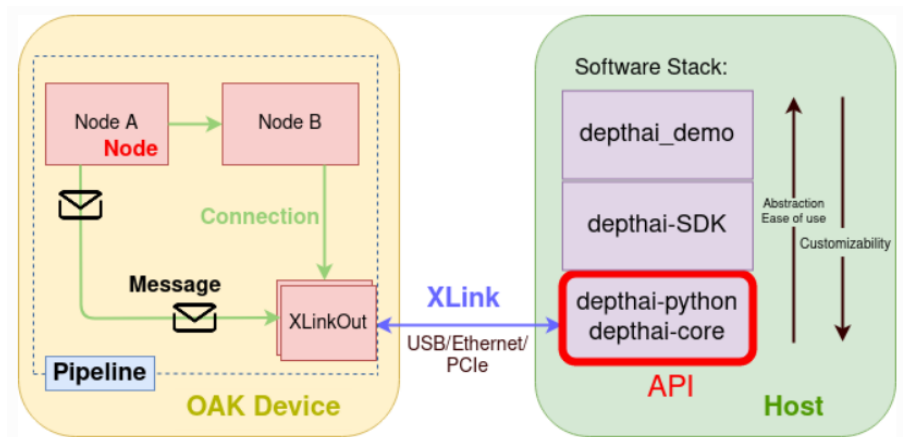


Figura 4.4: Arquitetura da API DepthAI [55]

Hospedeiro

Pode ser um computador, como um PC ou *Remote Programming Interface* (RPI), ao qual está ligado um dispositivo OAK [55].

Dispositivo

É o próprio dispositivo OAK. Se algo está a acontecer no lado do dispositivo, significa que está a funcionar na Intel Myriad X (UPV) [55].

Nós

Um nó é uma função que aplica algumas operações num determinado conjunto de entradas e produz um conjunto de saídas. Estes nós podem ser ligados de forma a que a saída de um nó seja utilizada como uma entrada por outro nó na *pipeline* [56].

Nos dois próximos pontos, vão ser exemplificados os dois nós mais básicos, que permitem que o computador hospedeiro interaja com o dispositivo OAK [56]:

- **XLinkIn** - O nó XLinkIn, mostrado na Figura 4.5, é utilizado para enviar dados do computador anfitrião para o dispositivo [56].

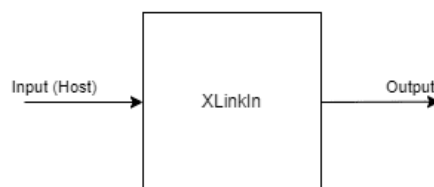


Figura 4.5: Nó XLinkIn [56]

Qualquer entrada pode ser enviada do lado do hospedeiro para o dispositivo [56].

A saída do nó é a entrada inalterada do computador hospedeiro que pode ser utilizada na *pipeline* [56]. O Código 4.7 mostra como pode ser criado o nó XLinkIn em Python.

Código 4.7: Código XLinkIn Python [56]

```
1 import depthai as dai
2
3 pipeline = dai.Pipeline()
4 xlinkIn = pipeline.create(dai.node.XLinkIn)
```

- **XLinkOut** - O nó XLinkOut, mostrado na Figura 4.6, é utilizado para enviar dados do dispositivo para o computador hospedeiro [56].

Qualquer entrada pode ser enviada do dispositivo para o lado do hospedeiro [56].

A saída do nó é a entrada inalterada do computador do dispositivo que pode ser utilizada pelo computador do hospedeiro [56]. O Código 4.8 mostra como pode ser criado o nó XLinkOut em Python.

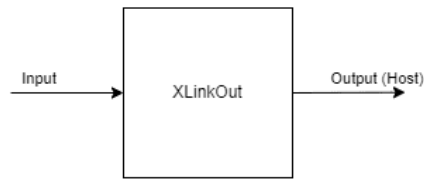


Figura 4.6: Nó XLinkOut [56]

Código 4.8: Código XLinkOut Python [56]

```
1 | import depthai as dai
2 |
3 | pipeline = dai.Pipeline()
4 | xlinkOut = pipeline.create(dai.node.XLinkOut)
```

XLink

É um *middleware* que é responsável pela troca de dados entre o dispositivo e o hospedeiro. O nó XLinkIn permite enviar os dados do hospedeiro para um dispositivo, enquanto que o XLinkOut faz o contrário [56].

Mensagens

Estas são transferidas entre nós, tal como definido por uma conexão [55].

Conexão

É uma ligação entre a saída de um nó e a entrada de outro. A fim de definir o fluxo de dados da *pipeline*, as conexões definem para onde enviar mensagens a fim de alcançar um resultado esperado [55].

Pipeline

Qualquer ação do DepthAI, quer seja uma inferência neural ou uma saída de câmara a cores, requer a definição de uma *pipeline*, incluindo nós e ligações correspondentes às necessidades [56].

Na *pipeline*, os diferentes nós estão ligados entre si, e os dados transitam de um nó para outro numa sequência definida, enquanto que são realizadas diferentes operações com os dados em cada nó [56].

Na Figura 4.7 está uma *pipeline* simples para obter a imagem RGB na câmara no OAK-D.

A *pipeline* é primeiro criada no hospedeiro, definindo as ligações entre os nós e o fluxo de dados. A *pipeline* definida é então enviada para o dispositivo OAK durante a inicialização e é executada com as entradas necessárias da câmara e de outros módulos, e a saída é enviada de volta para o hospedeiro [56].

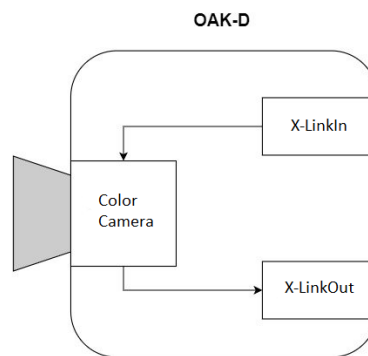


Figura 4.7: Pipeline para obter a imagem RGB da câmara [56]

4.2.4 Sockets

Um *socket* é um mecanismo de comunicação, usado normalmente para executar um modelo cliente/servidor, que permite a troca de mensagens entre os processos de uma máquina/aplicação servidor e de uma máquina/aplicação cliente [57].

Os *sockets* surgiram originalmente no sistema operativo Unix Berkeley Software Distribution (BSD). O BSD é um Sistema Operativo UNIX desenvolvido pela Universidade de Berkeley, na Califórnia, durante os anos 70 e 80 [57].

Um *socket* é usado para comunicar com um outro que esteja numa máquina qualquer. Este pode ser gerado como um portão de entrada e de saída de cada processo, podendo enviar e/ou receber mensagens através dos seus *sockets*. Devido ao facto que, usualmente, os protocolos de transporte da rede são implementados e controlados pelo sistema operativo, um *socket* também pode ser concebido como uma interface entre a aplicação e o substrato de comunicação do sistema operativo [57].

Existem dois *sockets* que são os mais comuns e utilizados [57]:

- **TCP** - *Transmission Control Protocol* (TCP) (*Stream Sockets*) é um dos principais protocolos de envio e recebimento de dados através da internet. O TCP é o protocolo mais usado, isto porque fornece garantia na entrega de todos os pacotes entre um PC emissor e um PC recetor [57].
- **UDP** - *User Datagram Protocol* (UDP) (*Datagram Sockets*) é um protocolo de comunicação que oferece uma quantidade limitada de serviço quando as mensagens são trocadas entre computadores numa rede que usa IP. Este é um protocolo mais simples e, por si só, não fornece garantia na entrega dos pacotes [57].

A Tabela 4.4 mostra as principais diferenças entre os protocolos TCP e UDP.

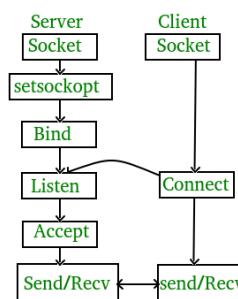
Tabela 4.4: Diferenças entre os protocolos TCP e UDP [57]

TCP	UDP
Orientado à conexão	Não necessita de conexão
Controlo de erro	Não possui controlo de erro
Garantia de entrega	Mais simples e rápido
Entrega ordenada	

Funções mais utilizadas nos *sockets* [57]:

- *Socket()* - cria um *socket* usado para comunicação e retorna um descritor;
- *Send()* - para enviar dados através de uma conexão TCP;
- *Read()* - para receber dados através de uma conexão TCP;
- *Close()* - para deslocar o *socket*;
- *Bind()* - usado para servidores para especificar uma porta na qual ele irá esperar conexões;
- *Listen()* - os servidores utilizam esta função para colocar o *socket* no modo passivo e torná-lo disponível para aceitar conexões;
- *Connect()* - depois de criar um *socket*, um cliente chama a função *Connect()* para estabelecer uma conexão com um servidor, usando o descritor do *socket*;
- *Accept()* - depois de um servidor chamar a função *Socket()* para criar um *socket*, *Bind()* para especificar o seu endereço e *Listen()* para colocá-lo no modo passivo, ele deve chamar o *Accept()* para pegar a primeira solicitação de conexão na fila.

A Figura 4.8 mostra um exemplo da comunicação entre dois *sockets*.

Figura 4.8: Exemplo de utilização de *sockets* [58]

4.2.5 RTDE

A interface RTDE fornece uma forma de sincronizar aplicações externas com o controlador UR sobre uma ligação TCP/IP padrão, sem quebrar quaisquer propriedades em tempo real do controlador Universal Robot (UR). Esta funcionalidade é útil, entre outras, para interagir com controladores de *fieldbus* (por exemplo, Ethernet/IP), manipular as Entradas/Saídas (E/S) do robô e traçar o estado do robô (por exemplo, trajetórias do robô). A interface RTDE está, por omissão, disponível quando o controlador UR está em execução [59].

A sincronização é configurável e pode, por exemplo, envolver os seguintes dados [59]:

- saída - estado do robô, juntas, ferramentas e segurança, E/S analógicas e digitais e registos de saída de uso geral;
- entrada - saídas digitais e analógicas e registos de entrada de uso geral.

A funcionalidade RTDE é dividida em duas fases: um procedimento de configuração e um ciclo de sincronização.

Ao ligar à interface RTDE, o cliente é responsável pela configuração das variáveis a sincronizar. Qualquer combinação de registos de entrada e saída, no controlador UR, que o cliente precise de escrever e ler, respetivamente, pode ser especificada. Para atingir este objetivo, o cliente envia uma lista de configuração dos campos de entrada e saída nomeados que devem estar contidos nos pacotes de sincronização de dados reais. A definição de um formato de pacote de sincronização de dados é referida como uma receita. Há um limite máximo de 2048 bytes para representar a lista de nomes de campos de entrada/saída que um cliente gostaria de subscrever [59].

Em troca, a interface RTDE responde com uma lista dos tipos de variáveis ou especifica que uma variável específica não foi encontrada. Cada receita de entrada que tenha sido configurada com sucesso obterá uma identificação de receita única. Quando a configuração estiver completa, a sincronização de dados pode ser iniciada e pausada [59].

Quando o ciclo de sincronização é iniciado, a interface RTDE envia ao cliente os dados solicitados na mesma ordem em que foram solicitados pelo cliente. Além disso, espera-se que o cliente envie entradas atualizadas para a interface RTDE em caso de alteração de valor. A sincronização de dados utiliza dados binários em série [59].

Todos os pacotes partilham a mesma estrutura geral, com um cabeçalho e uma carga útil, se aplicável. Os pacotes utilizados para o procedimento de configuração geram uma mensagem de retorno. Os pacotes do ciclo de sincronização não o fazem. Tanto o cliente como o servidor podem, a qualquer momento, enviar uma mensagem de texto, em que

o nível de aviso especifica a gravidade do problema. O RTDE está disponível na porta número 30004 [59].

Algumas características da interface RTDE [59]:

- sincronização em tempo real - a RTDE, geralmente, produz mensagens de saída a 125 Hz. No entanto, o ciclo em tempo real no controlador tem uma prioridade mais elevada do que a interface RTDE. Por conseguinte, se o controlador não tiver recursos computacionais, saltará uma série de pacotes de saída; o controlador enviará sempre os dados mais recentes e os pacotes ignorados não serão enviados mais tarde. Os pacotes de entrada serão sempre processados no ciclo de controlo que recebem, pelo que a carga para o controlador variará dependendo do número de pacotes recebidos [59];
- mensagens de entrada - a atualização das variáveis no controlador pode ser dividida em múltiplas mensagens. Pode-se ter uma mensagem para atualizar tudo, uma mensagem por variável, ou qualquer divisão entre elas. Não há necessidade de uma taxa de atualização constante; as entradas mantêm o seu último valor recebido. Apenas um cliente RTDE é autorizado a controlar uma variável específica em qualquer altura [59];
- ambiente de tempo de execução - um cliente RTDE pode correr no PC do controlador UR ou em qualquer PC externo. A vantagem de executar o cliente RTDE no controlador é a não existência de latência da rede. No entanto, o cliente RTDE e o controlador UR irão competir por recursos computacionais. O cliente RTDE deve correr com prioridade no sistema operativo padrão. Os processos computacionalmente intensivos, como, por exemplo, processamento de imagem, devem ser executados num PC externo [59];
- alterações de protocolo - o protocolo RTDE pode ser atualizado em qualquer altura pelo UR. Para garantir a máxima compatibilidade para o seu cliente RTDE, os clientes RTDE podem solicitar a interface RTDE para comunicar uma versão específica do protocolo. As adições/alterações de protocolo são explicitamente assinaladas, caso contrário assume-se a primeira versão [59].

4.3 Conclusão do capítulo 4

Neste capítulo foi apresentado e descrito todo o *hardware* utilizado neste projeto e as interfaces utilizadas para completar os objetivos propostos.

Capítulo 5

Implementação

Este capítulo descreve os desenvolvimentos realizados na resposta aos desafios e objetivos definidos. Não ser apresentadas todas as etapas realizadas para chegar à solução final, a plataforma de auscultação autónoma.

5.1 Avaliação da profundidade obtida pela câmera

Para avaliar a capacidade da câmera calcular a distância entre as costas da pessoa e o braço robótico, foram analisados os dados obtidos através da mesma, principalmente os que dizem respeito ao cálculo em profundidade, de forma a, com apenas essa informação, ser possível colocar o estetoscópio nos pontos corretos.

Foram então realizados uma série de testes, onde, primeiramente, foi posicionado um objeto alvo em diferentes distâncias conhecidas, e, de seguida, este mesmo objeto foi colocado a uma distância conhecida, e ao mesmo tempo foi colocado um outro objeto idêntico ao lado também em distâncias conhecidas. Para avaliar melhor as medidas recebidas, foi criado um gráfico *boxplot* para cada teste, onde foram utilizadas 20 medidas obtidas pela câmera.

As coordenadas das posições dos objetos (x,y,z) são calculadas através do algoritmo *HostSpatialCalc* fornecido no *github* da Luxonis [60]. Este algoritmo calcula as coordenadas espaciais de uma *Region-Of-Interest* (ROI) ou *Point-Of-Interest* (POI) com base na imagem de profundidade, calculando a média dos valores de profundidade na ROI, e removendo os valores fora do intervalo. Para utilizar um POI neste algoritmo é necessário criar uma ROI através deste POI, a mais pequena possível. O autor deste algoritmo alerta para a utilização de uma ROI muito pequena, dizendo para utilizar pelo menos uma ROI com 10x10 pixels de profundidade [60]. Nos testes que foram realizados apenas se teve em atenção o valor da coordenada espacial z , ou seja, a que distância o objeto está.

O *frame* RGB, mostrado no lado direito nas Figuras 5.2, 5.4, 5.6, 5.8, 5.10, 5.12, vindo da câmera colorida presente no *kit* OAK-D, foi obtido através da *pipeline* descrita, anteriormente, na Figura 4.7. Esta câmera foi utilizada com uma resolução 1080p, e um tamanho de imagem 640x360 pixels.

Para calcular as coordenadas espaciais, presentes nas Figuras 5.2, 5.4, 5.6, 5.8, 5.10, 5.12, do lado esquerdo no *frame* de disparidade, foram utilizadas as câmeras mono direita e esquerda, com a *pipeline* descrita na Figura 5.1. O par de câmeras mono foi utilizado com uma resolução de 400p e este está sintonizado em profundidade com a câmera colorida.

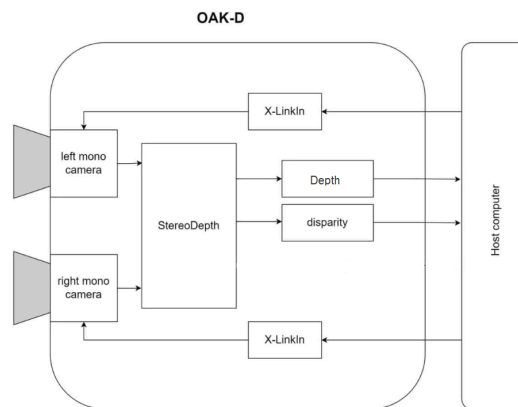


Figura 5.1: *Pipeline* para obter a imagem de profundidade e outra de disparidade.

Nesta *pipeline* são obtidas duas imagens, uma correspondente à profundidade e outra à disparidade. A imagem de profundidade é utilizada para calcular as coordenadas espaciais.

No primeiro teste realizado, o objeto alvo foi colocado a uma distância de 600 mm da câmera, como se pode ver pela Figura 5.2. Esta figura, e todas as outras idênticas a esta, correspondem a *frames* retiradas no momento em que se estava a recolher medidas para inserir nos respetivos gráficos *boxplot*.

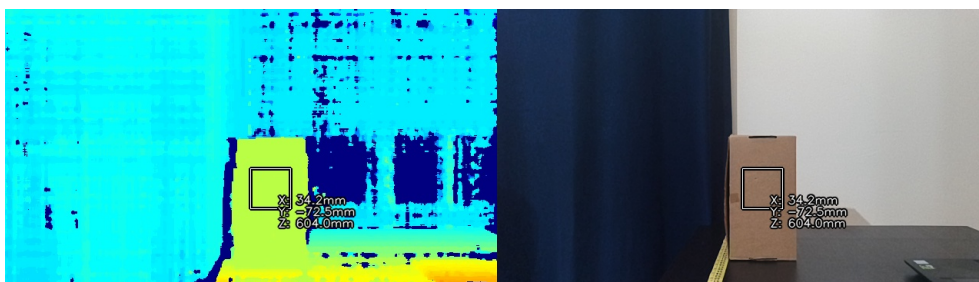


Figura 5.2: Teste com objeto alvo a 600 mm

Pelo gráfico *boxplot* mostrado na Figura 5.3, e obtido a partir dos resultados apresen-

tados na Figura 5.2, é possível verificar que este teste obteve uma mediana de 604 mm. Isto quer dizer que, o cálculo da profundidade errou, aproximadamente, 4 mm.

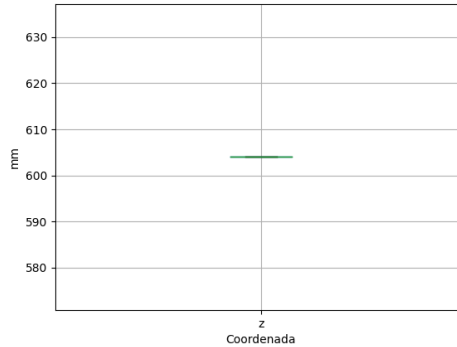


Figura 5.3: *Boxplot* da coordenada z com o objeto a 600 mm

No segundo teste realizado, o objeto alvo foi colocado a uma distância de 700 mm da câmera, como se pode ver pela Figura 5.4.

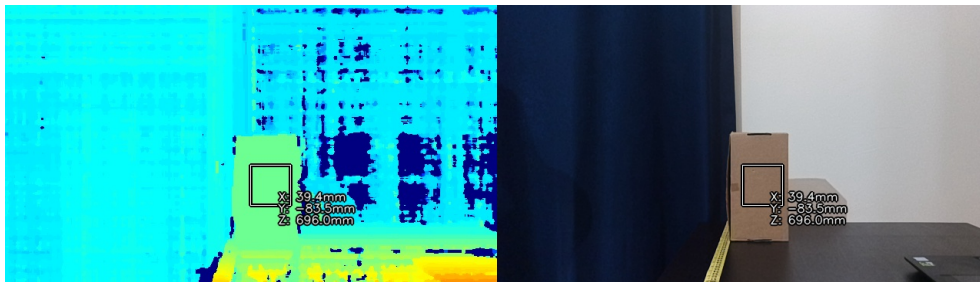


Figura 5.4: Teste com objeto a 700 mm

Pelo gráfico *boxplot* mostrado na Figura 5.5, e obtido a partir dos resultados apresentados na Figura 5.4, é possível verificar que este teste obteve uma mediana de 696 mm. Isto quer dizer que, tal como no teste anterior, o cálculo da profundidade errou em, aproximadamente, 4 mm.

No último teste realizado com apenas o objeto alvo, este foi colocado a uma distância de 800 mm, como se pode ver pela Figura 5.6.

Pelo gráfico *boxplot* mostrado na Figura 5.7, e obtido a partir dos resultados apresentados na Figura 5.6, é possível verificar que este teste obteve uma mediana de 801 mm. Isto quer dizer que, neste teste, o cálculo da profundidade errou em, aproximadamente, 1 mm.

Uma vez que a câmera terá que calcular as coordenadas espaciais para oito diferentes pontos de auscultação, foram realizados mais três novos testes. Estes foram realizados para analisar a influência nos resultados obtidos pela câmera ao adicionar um novo objeto

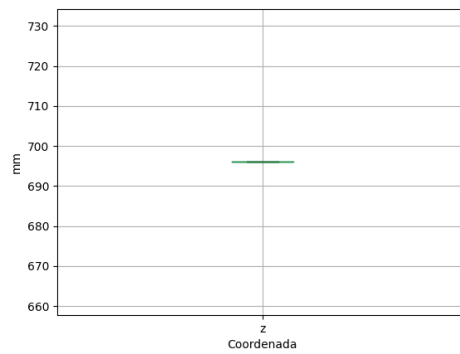


Figura 5.5: *Boxplot* da coordenada z com o objeto a 700 mm

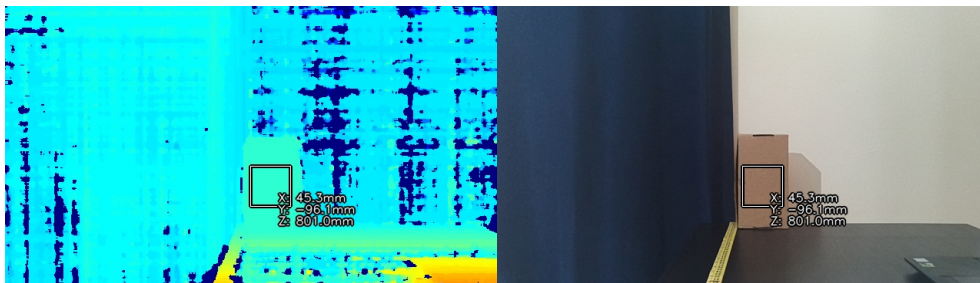


Figura 5.6: Teste com objeto a 800 mm

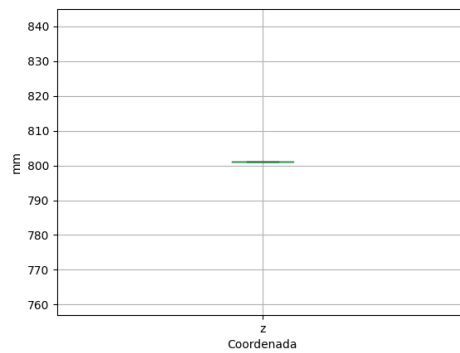


Figura 5.7: *Boxplot* da coordenada z com o objeto a 800 mm

idêntico. O objeto alvo foi colocado a uma distância de 700 mm em todos os novos testes.

Neste novo primeiro teste, o objeto idêntico foi colocado a 600 mm da câmera, como se pode ver pela Figura 5.8.

Pelo gráfico *boxplot* mostrado na Figura 5.9, e obtido a partir dos resultados apresen-

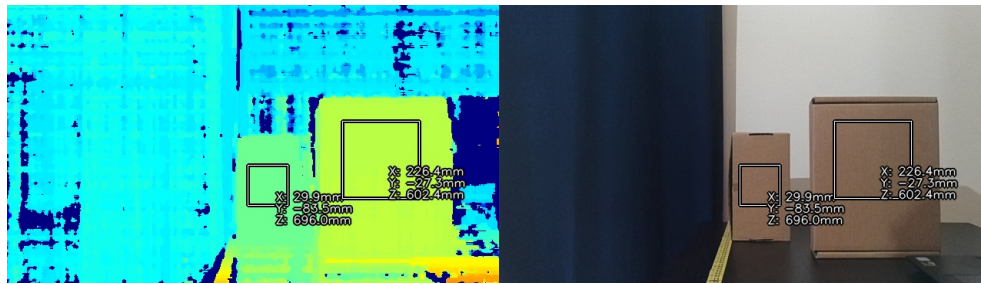


Figura 5.8: Teste com objeto alvo a 700 mm e objeto idêntico a 600 mm

tados na Figura 5.8, é possível verificar que este teste obteve uma mediana de 696 mm. Isto quer dizer que, neste teste, o cálculo da profundidade errou em, aproximadamente, 4 mm, tal como no teste em que só existia o objeto alvo, ou seja, a adição de um outro objeto idêntico neste teste não alterou o resultado. Quanto ao objeto idêntico, pode-se ver que possui um erro de aproximadamente 2 mm.

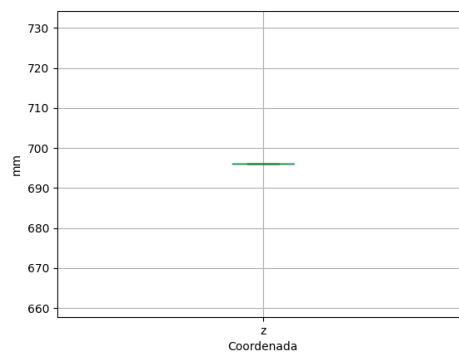


Figura 5.9: *Boxplot* da coordenada z do objeto alvo a 700 mm e com um objeto idêntico a 600 mm

No segundo novo teste, o objeto idêntico foi colocado à mesma distância que o objeto alvo, 700 mm, como se pode ver pela Figura 5.10.

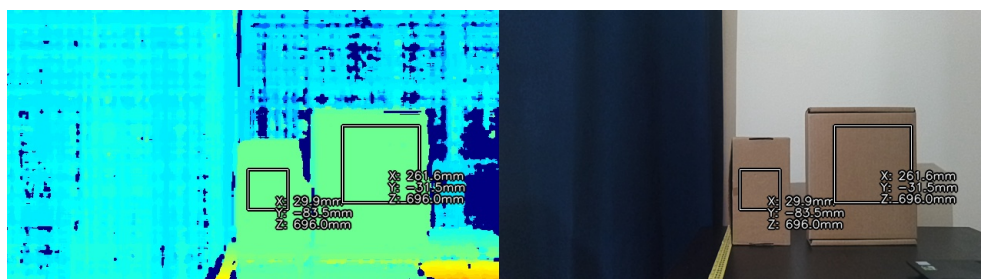


Figura 5.10: Teste com objeto alvo e objeto idêntico a 700 mm

Pelo gráfico *boxplot* mostrado na Figura 5.11, e obtido a partir dos resultados apresentados na Figura 5.10, é possível verificar que este teste obteve uma mediana de 696 mm. Pode-se ver também, pela Figura 5.10, que ambos os objetos estão à mesma distância calculada de 696 mm. Isto quer dizer que, neste teste, o cálculo da profundidade errou em, aproximadamente, 4 mm, não existindo alterações em relação ao teste onde apenas o objeto alvo foi colocado a uma posição de 700 mm.

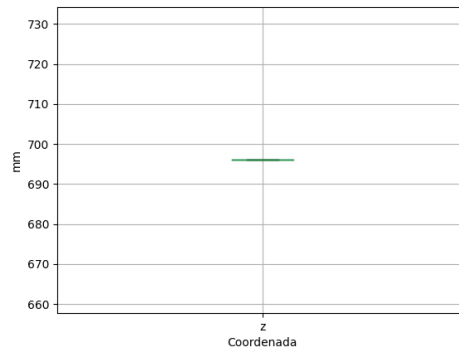


Figura 5.11: *Boxplot* da coordenada z do objeto alvo a 700 mm e com um objeto idêntico a 700 mm

No último teste, o objeto idêntico foi colocado a uma distância de 800 mm, como se pode ver pela Figura 5.12.

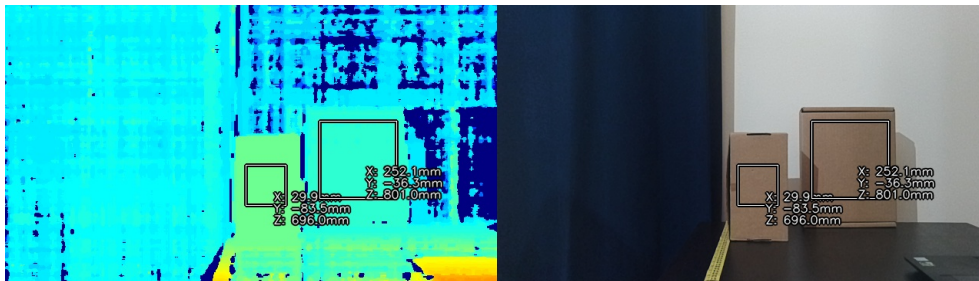


Figura 5.12: Teste com objeto alvo a 700 mm e objeto idêntico a 800 mm

Pelo gráfico *boxplot* mostrado na Figura 5.13, e obtido a partir dos resultados apresentados na Figura 5.12, é possível verificar que, mais uma vez, o valor da distância do objeto alvo em relação à câmera se manteve em 696 mm. Ou seja, o cálculo da profundidade errou novamente em, aproximadamente, 4 mm. Quanto ao objeto idêntico pode-se ver que este está a uma distância calculada de 801 mm, coincidindo com o valor mostrado na Figura 5.6, quando o objeto alvo foi colocado a uma distância de 800 mm.

Realizados estes testes é possível concluir que, em geral, esta câmera possui bons resultados, uma vez que o erro máximo existente nestes testes é de 4 mm. Apesar disso, este projeto não permite qualquer margem de erro quando calculada a que distância as

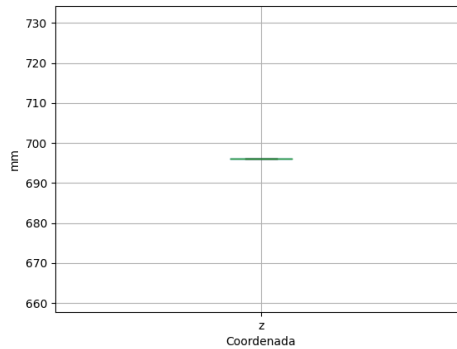


Figura 5.13: *Boxplot* da coordenada z do objeto alvo a 700 mm e com um objeto idêntico a 800 mm

costas do paciente estão do braço robótico, ou seja, ter uma margem de 4 mm de erro pode corresponder ao estetoscópio estar devidamente encostado nas costas da pessoa, ou não. Dito isto, vai ser utilizada uma outra abordagem descrita mais à frente.

5.2 Treino do modelo de previsão para identificar os pontos a auscultar

Para identificar os pontos a auscultar nas costas da pessoa foram necessárias duas abordagens para treinar um modelo de previsão, uma vez que na primeira não se obteve bons resultados.

5.2.1 Primeira abordagem

Nesta abordagem foi treinado um modelo para prever diretamente os pontos a auscultar nas costas da pessoa. Para isso foi necessário criar um *Dataset* composto por uma pasta de treino e uma de validação. Dentro destas pastas estão imagens de costas e as suas respetivas máscaras, ambas com um tamanho de 512x512x3 pixels. Na Figura 5.14 é possível ver um pequeno exemplo de uma imagem de costas com a sua respetiva máscara, ou seja, os pontos a auscultar, como foi visto na Figura 1.1.

O *Dataset* sofreu várias alterações, como é possível ver pela Tabela 5.1.

Estas alterações foram realizadas para verificar a sua influência no treino do modelo:

- as técnicas de aumento de dados foram alternadas entre todas as exemplificadas na Figura 5.15 ou apenas quatro delas (*HorizontalFlip*, *VerticalFlip*, *Transpose* e *RandomRotate90*);

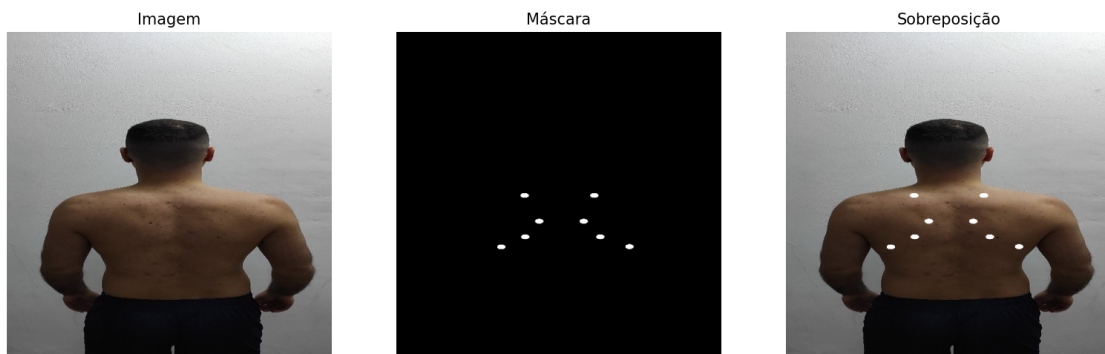


Figura 5.14: Exemplo de uma imagem de costas e a sua respetiva máscara, presentes no *Dataset*

Tabela 5.1: Mudanças no treino do *Dataset* e no respetivo *Dataset*

Modelo/Tentativa	α	Aumento Dados	Imgs Net	Nº imgs treinar	Nº imgs validar
MobileNetV2_1	1,4	Todas	Não	550	11
MobileNetV2_2	1,4	Todas	Sim	630	11
MobileNetV2_3	0,5	4	Não	275	11
MobileNetV2_4	0,5	Todas	Sim	630	11
MobileNetV2_5	1,4	4	Sim	315	55
MobileNetV2_6	1,4	4	Sim	315	11
MobileNetV2_7	1,4	4	Não	275	11
MobileNetV2*	1,4	Todas	Sim	1190	14
InceptionResNetV2_1	X	Todas	Sim	630	11
InceptionResNetV2_2	X	Todas	Sim	1220	11
InceptionResNetV2_3	X	4	Sim	610	11
InceptionResNetV2*	X	4	Sim	610	14

- foram aplicadas as quatro técnicas de aumento de dados acima descritas na pasta de validação, aumentando o número das imagens presentes;
- foram adicionadas imagens de costas de pessoas encontradas na Internet;
- os modelos também foram alterados entre os dois exemplificados anteriormente e no modelo MobileNetV2-UNet também foi alterado o valor do α , fazendo com isto alterar o número de parâmetros de treino;
- nos dois modelos que possuem um asterisco no nome da tentativa foi realizada uma segmentação de imagem em humanos com um modelo de previsão já treinado e disponível na Internet [61]. Este modelo prevê uma máscara que corresponde ao contorno da pessoa, e com ela é possível realizar a extração do fundo da imagem. Na Figura 5.15, na imagem com a legenda extração de fundo, está uma exemplificação deste processo.

O modelo utilizado para a segmentação de imagem em humanos foi criado utilizando

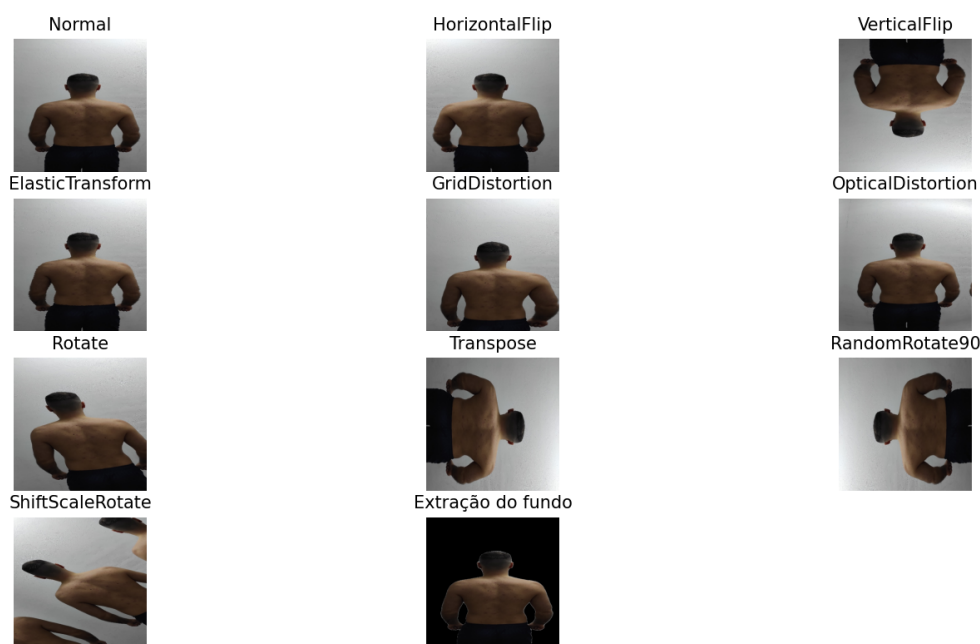


Figura 5.15: Técnicas de aumento de dados da primeira abordagem

uma arquitetura DeepLabV3+ e obteve os resultados demonstrados na Tabela 5.2 quando aplicado no *Dataset* de validação/teste em que foi treinado [61].

Tabela 5.2: Resultados do modelo utilizado para a segmentação de imagem em humanos quando aplicado no *Dataset* de validação/teste em que foi treinado [61].

<i>Accuracy</i>	<i>F1-score</i>	<i>Jaccard</i>	<i>Recall</i>	<i>Precision</i>
0,9740	0,954	0,921	0,959	0,957

Os modelos utilizados para treinar o *Dataset* foram o MobileNetV2-UNet e o Inception-ResNet-V2-UNet, descritos anteriormente neste trabalho. Nos Códigos C.1 e C.2, presentes no Anexo C, pode-se ver a criação dos modelos MobileNetV2-UNet e Inception-ResNet-V2-UNet em Python, respectivamente.

No modelo MobileNetV2-UNet ao utilizar a rede MobileNetV2 como codificadora é possível alterar o número de parâmetros da mesma através do argumento *alpha*. Para os dois seguintes valores de *alpha* (valor máximo e um dos valores mínimos, respectivamente):

- 1,4 possui:
 - 11 753 809 parâmetros totais;
 - 11 725 617 parâmetros de treino;

- 28 192 parâmetros não treináveis.
- 0,5 possui:
 - 8 450 673 parâmetros totais;
 - 8 437 985 parâmetros de treino;
 - 12 688 parâmetros não treináveis.

O modelo Inception-ResNet-V2-UNet possui uma rede neuronal com 36 793 089 parâmetros, em que 36 752 065 destes são de treino e 41 024 parâmetros não são treináveis.

Esta diferença de parâmetros entre estes dois modelos leva a que o treino e a previsão demore mais (se tiver mais parâmetros) ou menos tempo (se tiver menos parâmetros), como também já foi visto anteriormente neste trabalho.

Para compilar o modelo foi utilizado o otimizador Adam, com uma taxa de aprendizagem de 0,0001, foi utilizada a *dice loss* como função de perda, tanto no treino como na validação, e as métricas DSC, IoU, *Recall* e *Precision* para analisar o modelo ao longo do treino, como é possível ver pelo Código 4.3. Para testar o modelo são utilizadas as métricas *Accuracy*, *F1-score*, *Jaccard*, *Recall* e *Precision*.

Os modelos foram treinados ao longo de 40 *epochs*, com um *batch size* de 1, na GPU do computador descrito na subsecção 4.1.2 deste trabalho.

Para treinar o modelo foi utilizada a função *fit()* demonstrada no Código 4.4 e as *callbacks* utilizadas estão demonstradas no Código 5.1. Estas são apenas algumas indicações ou funções a aplicar no final de cada *epoch*.

Código 5.1: *Callbacks* utilizadas durante o treino

```

1 | callbacks = [
2 |     ModelCheckpoint(model_path, verbose=1, save_best_only=False
3 |         , mode='auto', save_freq='epoch'),
4 |     ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience
5 |         =5, min_lr=1e-6, verbose=1),
6 |     CSVLogger(csv_path),
7 |     TensorBoard(),
8 |     EarlyStopping(monitor="loss", patience=5,
9 |         restore_best_weights=False)
10 | ]

```

As *callbacks* mais importantes a reter do Código 5.1 são:

- *ModelCheckpoint()*: grava o modelo numa pasta no final de cada *epoch*;
- *ReduceLROnPlateau()*: reduz a taxa de aprendizagem quando a perda no *Dataset* de validação não desce há 5 *epochs*;

- *EarlyStopping()*: se o valor da função de perda no treino também não descer há 5 *epochs*, acaba o treino.

Os valores das métricas *Accuracy*, *F1-score*, *Jaccard*, *Recall* e *Precision* presentes na Tabela 5.3 são da *epoch* em que o modelo obteve o menor valor na função de perda no *Dataset* de validação em 40 *epochs*. Os valores da coluna *Loss* correspondem ao valor da função de perda no *Dataset* de treino no final de 40 *epochs*. Para avaliar o modelo, ou prever os pontos numa nova imagem através deste mesmo modelo, é utilizada a função *predict()*, demonstrada anteriormente no excerto de Código 4.5. Tendo em conta esta tabela, os resultados dos treinos realizados não são suficientes para este trabalho. Os valores destacados a verde são os melhores por métrica.

Tabela 5.3: Resultados do treino, em concordância com a Tabela 5.2

Modelo/Tentativa	<i>Val_loss</i>	<i>Loss</i>	<i>Accuracy</i>	<i>F1-score</i>	<i>Jaccard</i>	<i>Recall</i>	<i>Precision</i>
MobileNetV2_1	0,783	0,136	0,9976	0,168	0,094	0,192	0,157
MobileNetV2_2	0,702	0,171	0,9962	0,217	0,122	0,380	0,154
MobileNetV2_3	0,786	0,186	0,9975	0,165	0,092	0,196	0,156
MobileNetV2_4	0,729	0,269	0,9964	0,191	0,107	0,333	0,144
MobileNetV2_5	0,843	0,141	0,9977	0,122	0,066	0,123	0,130
MobileNetV2_6	0,776	0,125	0,9977	0,174	0,097	0,180	0,173
MobileNetV2_7	0,770	0,117	0,9969	0,166	0,092	0,236	0,122
MobileNetV2*	0,688	0,225	0,9975	0,158	0,087	0,189	0,145
InceptionResNetV2_1	0,693	0,131	0,9966	0,221	0,125	0,356	0,163
InceptionResNetV2_2	0,662	0,134	0,9968	0,242	0,139	0,370	0,180
InceptionResNetV2_3	0,726	0,127	0,9970	0,205	0,115	0,294	0,161
InceptionResNetV2*	0,646	0,135	0,9980	0,266	0,165	0,328	0,264

A tentativa InceptionResNetV2* obteve os melhores valores, mas os resultados ainda apresentavam uma perda significativa, o que levava a que não encontrasse alguns dos pontos de auscultação desejados. Uma vez que os pontos não encontrados eram aleatórios, tornava-se difícil prever quais pontos estavam em falta e de que forma prever estes indiretamente. Estes resultados podem não ter sido satisfatórios devido à criação das máscaras, uma vez que os pontos colocados nas costas da mesma pessoa em imagens diferentes nunca correspondem ao mesmo lugar em todas as imagens, tal como o prestador de cuidados médicos dificilmente ausculta a mesma pessoa em diferentes momentos utilizando exatamente os mesmos pontos de auscultação.

Deste modo, optou-se por uma segunda abordagem.

5.2.2 Segunda abordagem

Nesta segunda abordagem optou-se treinar um modelo para prever pontos de referência nas costas da pessoa, para através destes pontos calcular os pontos de auscultação. Na Figura 5.16 é possível ver onde são colocados estes pontos de referência numa imagem das costas de uma pessoa.

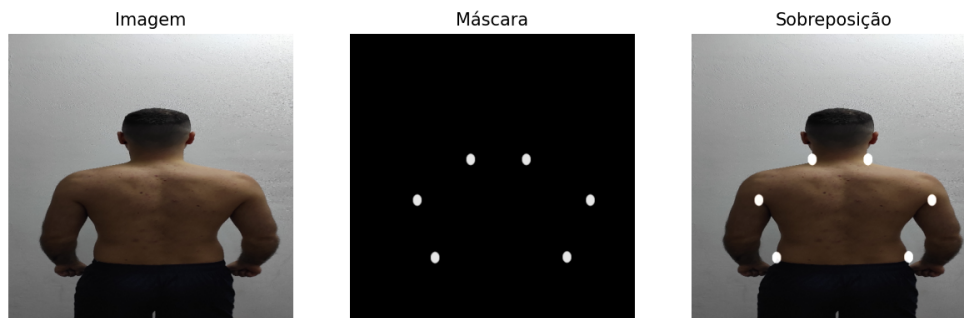


Figura 5.16: Exemplo de um par de imagens com os pontos de referência para treinar

O processo foi semelhante ao descrito na primeira abordagem. O *Dataset* tinha a mesma estrutura acima descrita, mas desta vez aumentou-se o número de imagens de costas recolhido de diferentes pessoas para realizar as técnicas de aumento de dados. E também se aumentou este mesmo número de técnicas de aumento de dados, descritas na Figura 5.17. A este *Dataset* foram adicionadas fotos de costas de pessoas do sexo feminino, ao contrário dos *Datasets* da abordagem anterior. Foram também utilizados os dois modelos já mencionados anteriormente. Nesta abordagem, como os resultados foram satisfatórios, não foi utilizada a extração do fundo através da segmentação de imagem de humanos para a previsão dos pontos de referência. A Tabela 5.4 apresenta algumas das características do *Dataset* utilizado. De notar que neste *Dataset* estavam presentes imagens de costas de 26 diferentes pessoas e 8 imagens retiradas da Internet.

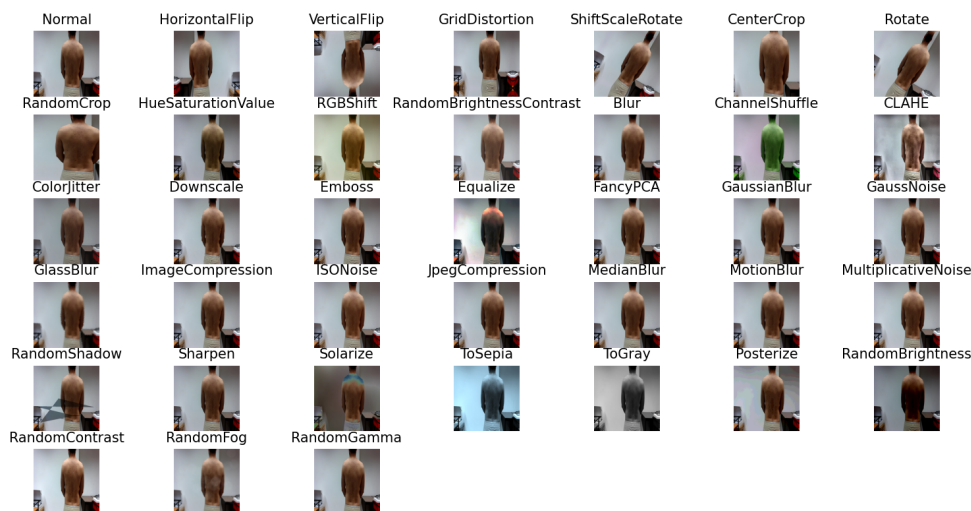


Figura 5.17: Técnicas utilizadas para o aumento dos dados

Os resultados obtidos no treino dos dois modelos foram muito semelhantes, como é possível ver pela Tabela 5.5. Desta forma, o modelo treinado escolhido para prever os pontos de referência nas costas de pessoas foi o MobileNetV2-UNet, uma vez que, ape-

Tabela 5.4: Características do *Dataset* utilizado para o treino da rede

Modelo	Alpha	Aumento de dados	Imgs Net	Nº imgs treinar	Nº imgs validar
MobileNetV2-UNet	1,4	37	Sim	6536	24
InceptionResNetV2-UNet	X	37	Sim	6536	24

sar da diferença ser pequena, apresenta melhores resultados do que o InceptionResNetV2-UNet e menos parâmetros, fazendo com que o modelo seja mais “leve”. Apesar do valor da função de perda no *Dataset* de validação estar em 0,338, o modelo apenas não prevê um ponto de referência em duas imagens das 24 utilizadas para validação, sendo estas 24 imagens de diferentes pessoas.

Tabela 5.5: Resultados do treino, em concordância com a Tabela 5.4

Modelo	Val_loss	Loss	Accuracy	F1-score	Jaccard	Recall	Precision
MobileNetV2_UNet	0,338	0,162	0,9973	0,713	0,557	0,654	0,805
InceptionResNetV2_UNet	0,340	0,154	0,9973	0,698	0,540	0,603	0,851

5.3 Auscultação autónoma

Obtido o modelo para prever os pontos de referência para realizar a auscultação nas costas das pessoas, passou-se para a construção do algoritmo para realizar a auscultação autónoma em tempo real. A Figura 5.18 mostra a arquitetura geral do projeto.

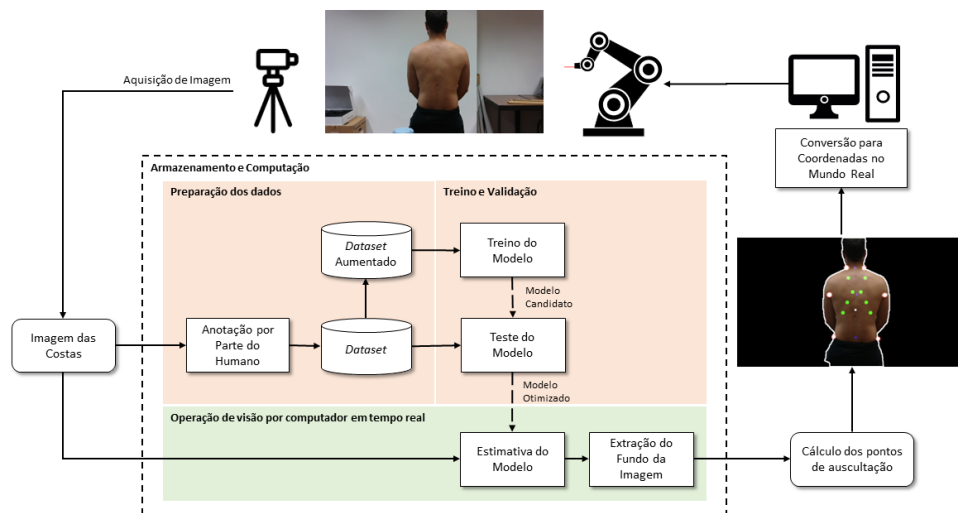


Figura 5.18: Arquitetura geral do projeto.

5.3.1 Cálculo do *offset* do alinhamento da câmera com o braço robótico

Para o centro da imagem obtida através da câmera corresponder ao centro do *gripper* do robô, e com isto os pontos espaciais de auscultação, calculados mais à frente, corresponderam às distâncias a que o braço robótico terá que ir, a câmera e o braço robótico terão que estar alinhados. Uma vez que a câmera não é fixa e que no fim de cada dia de testes é retirada da sua posição, dificilmente no próximo dia de testes esta câmera irá ser colocada na posição desejada, ou na mesma posição do dia anterior. Além disso, qualquer mínimo toque no suporte da câmera faz com que esta se mova. Deste modo, é calculado o *offset* do alinhamento da câmera com o braço robótico.

Para isso, foi criado um *socket* sendo o *host* o IP do braço robótico e utilizada a porta 30002. Através deste *socket* o braço robótico foi movimentado para as seguintes posições de junta, utilizando a função *movej* [62]:

- *base*: 180°;
- *shoulder*: -180°;
- *elbow*: 90°;
- *wrist 1*: -90°;
- *wrist 2*: 90°;
- *wrist 3*: 245°;

Este foi movimentado com uma velocidade de 2 m/s e uma aceleração de 2 m/s².

De seguida, a câmera é colocada em cima da mesa, de forma a que o ponto vermelho presente na Figura 5.19, que corresponde à imagem vinda da câmera colorida obtida através da *pipeline* descrita anteriormente na Figura 4.7, coincida aproximadamente com o ponto verde no centro do círculo verde, visto que este ponto verde coincide, aproximadamente, com o meio do *gripper* na parte frontal do braço robótico. Na primeira vez, foram colocadas marcações na mesa para a colocação do suporte da câmera, para da próxima vez haver uma ideia de onde o colocar, de forma à câmera ficar, aproximadamente, centrada com o robô.

Uma vez obtida a imagem da câmera colorida, esta é convertida para um formato *Hue, Saturation, Value* (HSV). O ponto vermelho é o centro da imagem. O círculo verde foi determinado através da função *HoughCircles()* que determinou os círculos presentes na imagem, tendo em conta os parâmetros inseridos na mesma, e os valores dos centros dos círculos encontrados presentes nos intervalos *Hue*[75:109], *Saturation*[63:238] e *Value*[0:255] correspondiam ao círculo desejado. Este círculo desejado é o verde e o seu centro é o ponto verde.



Figura 5.19: Cálculo do *offset* do alinhamento do braço robótico com a câmara

Para obter um valor mais preciso das coordenadas 2D do ponto verde, foi realizada uma média dos valores obtidos ao fim de 250 instâncias.

Tendo as coordenadas 2D do ponto verde, foram calculadas as suas coordenadas espaciais, como ilustrado na Figura 5.20. O valor destas coordenadas é o *offset*, isto porque, se este ponto verde estiver no centro da imagem, as coordenadas espaciais x e y vão dar 0.

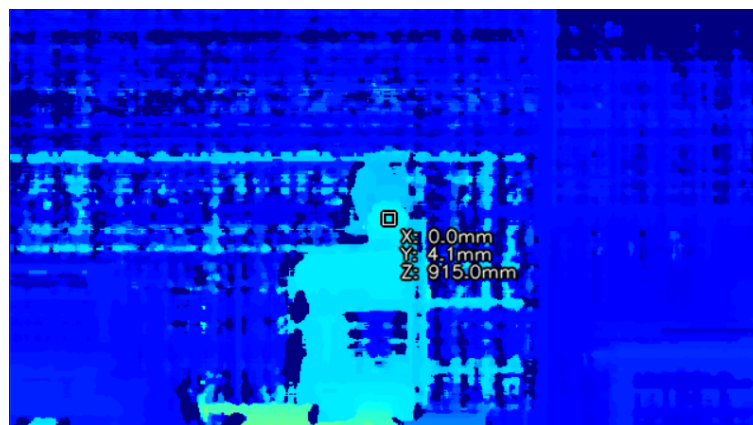


Figura 5.20: Cálculo das coordenadas espaciais do *offset*

Tendo por base o que foi descrito acima, foi então utilizada a função *calc_spatials()* do algoritmo *HostSpatialCalc*, presente no Código C.4 no Anexo C, onde foi inserida a imagem de profundidade e as coordenadas x e y do POI que se pretende calcular.

Para obter um valor mais preciso do *offset*, foi realizada também uma média dos valores obtidos ao fim de 250 instâncias.

5.3.2 Previsão dos pontos de referência

Calculado o *offset* do alinhamento da câmera com o braço robótico, procedeu-se à previsão dos pontos de referência. Através do *socket* anteriormente criado, o braço robótico foi movimentado para as seguintes posições de junta, utilizando a função *movej* [62]:

- *base*: 90°;
- *shoulder*: 0°;
- *elbow*: 0°;
- *wrist 1*: -90°;
- *wrist 2*: 90°;
- *wrist 3*: 240°;

Este movimento foi realizado com uma velocidade de 2 m/s e uma aceleração de 2 m/s².

Esta movimentação do braço robótico foi realizada para a câmera poder tirar um *frame* das costas da pessoa, sem o braço estar à frente das mesmas. A pessoa deve estar, aproximadamente, no centro da imagem.

Retirado o *frame* das costas da pessoa, como está ilustrado na Figura 5.21, este é redimensionado para 512x512x3 pixels, uma vez que esta é a entrada do modelo, como se pode ver pelo Código C.1 no Anexo C. Depois, foi utilizado o modelo para segmentação de imagem em humanos, já mencionado neste trabalho. A máscara prevista por este modelo (contorno da pessoa), demonstrada na Figura 5.22, vai ser utilizada mais à frente para calcular o centro deste mesmo contorno. De seguida, é feita a previsão dos pontos de referência, mostrados na Figura 5.23, através da função *predict()* presente no Código 4.5.

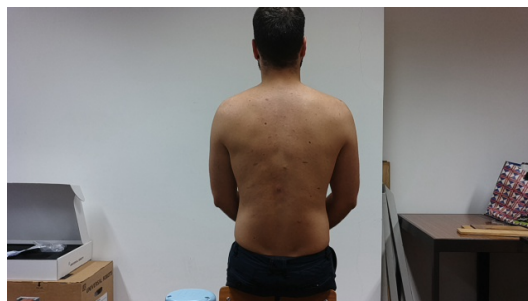


Figura 5.21: *Frame* das costas da pessoa



Figura 5.22: Extração de fundo do *frame* das costas da pessoa

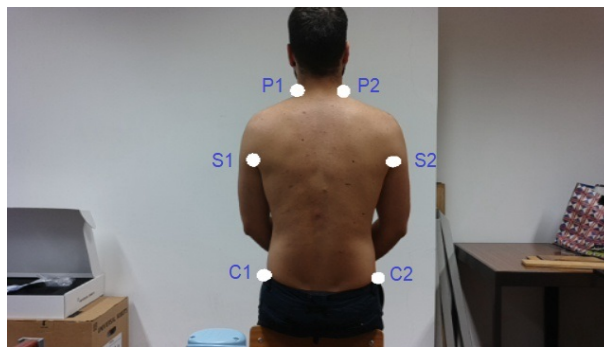


Figura 5.23: Pontos de referência nas costas da pessoa

5.3.3 Cálculo dos pontos a auscultar

Uma vez obtidos os pontos de referência nas costas da pessoa, passou-se ao cálculo dos pontos a auscultar. Para isso, utilizou-se a imagem prevista com os pontos de referência, a máscara com o contorno do corpo e o *frame* das costas da pessoa.

Primeiramente, foram retirados os pontos não desejados que foram erradamente previstos na imagem com os pontos de referência, ou seja, o ruído. Para isso foram utilizadas transformações morfológicas, nomeadamente *opening*. *Opening* é o nome dado a uma transformação de erosão seguido de dilatação. Estas transformações foram realizadas através da função *morphologyEx()*.

De seguida, foram obtidas as coordenadas 2D dos pontos de referência (*keypoints*) das costas da pessoa, através da função *SimpleBlobDetector* com os seguintes parâmetros:

- *filterByColor*: *True*;
- *blobColor*: 255;
- *minArea*: 0,1;

- *minCircularity*: 0,01;
- *minConvexity*: 0,01;
- *minThreshold*: 1;
- *minInertiaRatio*: 0,01.

Além disso, também foi obtido o centro do maior contorno presente na máscara com o contorno da pessoa. Este centro deve corresponder ao centro do contorno da pessoa.

Seguidamente, os *keypoints* são guardados em diferentes variáveis, uma vez que a função *SimpleBlobDetector* armazena todos estes pontos numa só variável. São também separados em lado direito ou lado esquerdo tendo em conta o centro anteriormente calculado. Estes pontos são depois também ordenados tendo em conta o valor da coordenada *y*.

Depois dos *keypoints* estarem todos ordenados e armazenados, se foram encontrados os *keypoints* (ver Figura 5.24):

- na zona da axila (*S1* e *S2*):
 - calcula-se o ponto intermédio entre os pontos *S1* e *S2* (*S_mid*), e o comprimento horizontal das costas através da coordenada *x* destes pontos (diferença entre estes dois pontos) (*dist_x*)
- na zona do pescoço (*P1*, *P2*):
 - calcula-se o ponto intermédio entre os pontos *P1* e *P2* (*P_mid*)
- na zona da cintura (*C1* e *C2*):
 - calcula-se o ponto intermédio entre os pontos *C1* e *C2* (*C_mid*)
- na zona do pescoço e da cintura (*P1*, *P2*, *C1* e *C2*):
 - calcula-se o comprimento vertical das costas (*dist_y*) através da diferença da coordenada *y* dos pontos intermédios calculados anteriormente (*P_mid*, *C_mid*)
- apenas no pescoço e na cintura do lado esquerdo (*P1* e *C1*):
 - calcula-se o comprimento vertical das costas através da coordenada *y* destes pontos *P1* e *C1* (diferença entre estes dois pontos) (*dist_y*)
- apenas no pescoço do lado direito e na cintura do lado esquerdo (*P2* e *C1*):
 - calcula-se o comprimento vertical das costas através da coordenada *y* destes pontos *P2* e *C1* (diferença entre estes dois pontos) (*dist_y*)

- apenas no pescoço do lado esquerdo e na cintura do lado direito ($P1$ e $C2$):
 - calcula-se o comprimento vertical das costas através da coordenada y destes pontos $P1$ e $C2$ (diferença entre estes dois pontos) ($dist_y$)
- apenas no pescoço e na cintura do lado direito ($P2$ e $C2$):
 - calcula-se o comprimento vertical das costas através da coordenada y destes pontos $P2$ e $C2$ (diferença entre estes dois pontos) ($dist_y$)

O algoritmo está construído de forma a que este só avance para a próxima etapa quando encontrar, pelo menos, 4 pontos de referência, onde obrigatoriamente tem que encontrar os pontos $S1$ e $S2$, porque é a única forma deste calcular o comprimento horizontal das costas. E tem de encontrar um ponto no pescoço ($P1$ ou $P2$) e outro no fundo das costas ($C1$ ou $C2$), para conseguir calcular o comprimento vertical das costas. Para calcular as coordenadas dos pontos 2D de auscultação, representados na Figura 5.24 a verde, através das medidas do comprimento das costas (vertical e horizontal), se estas forem obtidas com sucesso, e encontrados os *keypoints*:

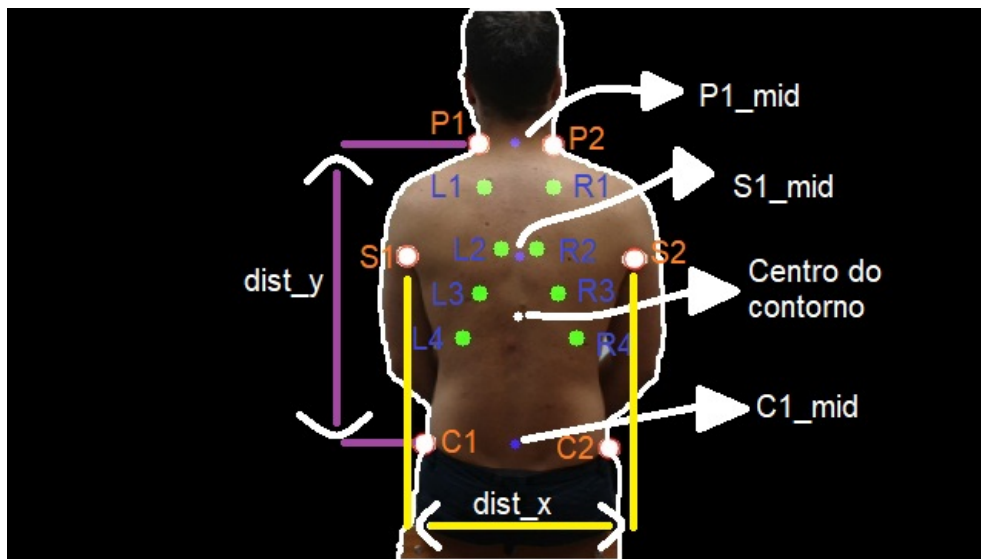


Figura 5.24: Pontos de auscultação 2D

- $P1$ e $P2$:
 - $L1_y = P_mid_y + dist_y * 0,15$ $L1_x = S_mid_x - (dist_x/2) * 0,30$
 - $R1_y = P_mid_y + dist_y * 0,15$ $R1_x = S_mid_x + (dist_x/2) * 0,30$
 - $L2_y = P_mid_y + dist_y * 0,35$ $L2_x = S_mid_x - (dist_x/2) * 0,15$

$$\begin{aligned}
- R2_y &= P_mid_y + dist_y * 0,35 & R2_x &= S_mid_x + (dist_x/2) * 0,15 \\
- L3_y &= P_mid_y + dist_y * 0,50 & L3_x &= S_mid_x - (dist_x/2) * 0,35 \\
- R3_y &= P_mid_y + dist_y * 0,50 & R3_x &= S_mid_x + (dist_x/2) * 0,35 \\
- L4_y &= P_mid_y + dist_y * 0,65 & L4_x &= S_mid_x - (dist_x/2) * 0,50 \\
- R4_y &= P_mid_y + dist_y * 0,65 & R4_x &= S_mid_x + (dist_x/2) * 0,50
\end{aligned}$$

- apenas $P1$:

$$\begin{aligned}
- L1_y &= P1_y + dist_y * 0,15 & L1_x &= S_mid_x - (dist_x/2) * 0,30 \\
- R1_y &= P1_y + dist_y * 0,15 & R1_x &= S_mid_x + (dist_x/2) * 0,30 \\
- L2_y &= P1_y + dist_y * 0,35 & L2_x &= S_mid_x - (dist_x/2) * 0,15 \\
- R2_y &= P1_y + dist_y * 0,35 & R2_x &= S_mid_x + (dist_x/2) * 0,15 \\
- L3_y &= P1_y + dist_y * 0,50 & L3_x &= S_mid_x - (dist_x/2) * 0,35 \\
- R3_y &= P1_y + dist_y * 0,50 & R3_x &= S_mid_x + (dist_x/2) * 0,35 \\
- L4_y &= P1_y + dist_y * 0,65 & L4_x &= S_mid_x - (dist_x/2) * 0,50 \\
- R4_y &= P1_y + dist_y * 0,65 & R4_x &= S_mid_x + (dist_x/2) * 0,50
\end{aligned}$$

- apenas $P2$:

$$\begin{aligned}
- L1_y &= P2_y + dist_y * 0,15 & L1_x &= S_mid_x - (dist_x/2) * 0,30 \\
- R1_y &= P2_y + dist_y * 0,15 & R1_x &= S_mid_x + (dist_x/2) * 0,30 \\
- L2_y &= P2_y + dist_y * 0,35 & L2_x &= S_mid_x - (dist_x/2) * 0,15 \\
- R2_y &= P2_y + dist_y * 0,35 & R2_x &= S_mid_x + (dist_x/2) * 0,15 \\
- L3_y &= P2_y + dist_y * 0,50 & L3_x &= S_mid_x - (dist_x/2) * 0,35 \\
- R3_y &= P2_y + dist_y * 0,50 & R3_x &= S_mid_x + (dist_x/2) * 0,35 \\
- L4_y &= P2_y + dist_y * 0,65 & L4_x &= S_mid_x - (dist_x/2) * 0,50 \\
- R4_y &= P2_y + dist_y * 0,65 & R4_x &= S_mid_x + (dist_x/2) * 0,50
\end{aligned}$$

Este algoritmo está descrito de forma mais direta e sucinta nos fluxogramas das Figuras 5.25, 5.26.

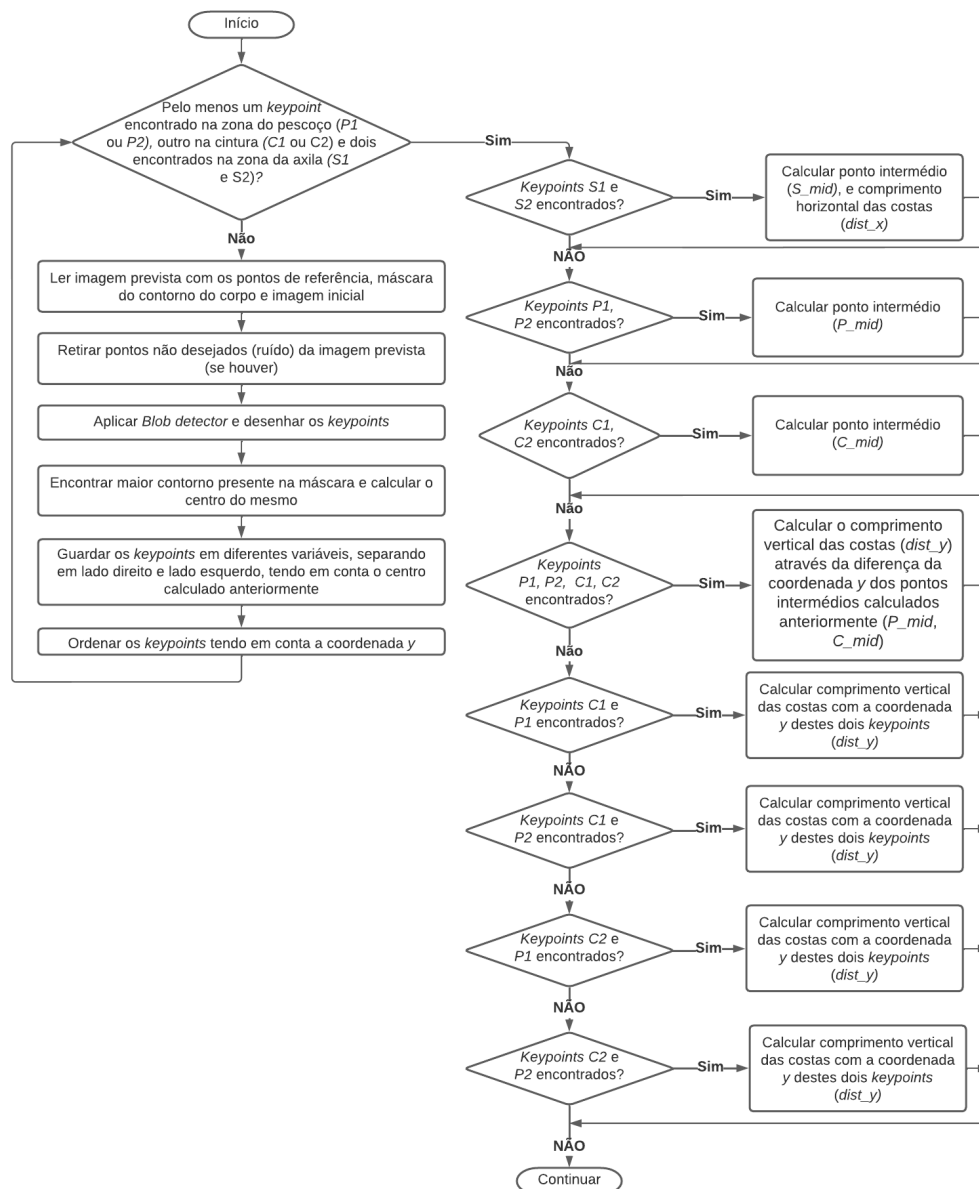


Figura 5.25: Primeira parte do algoritmo de cálculo dos pontos 2D de auscultação

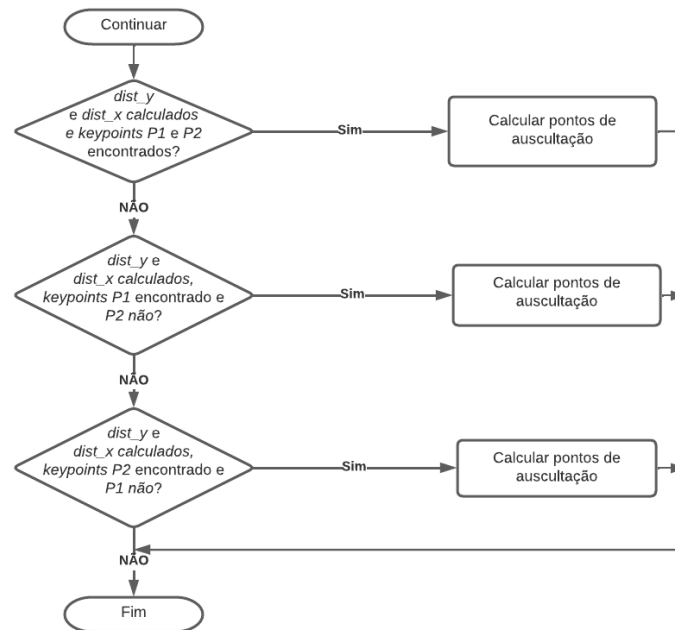


Figura 5.26: Segunda parte do algoritmo de cálculo dos pontos 2D de auscultação

5.3.4 Cálculo das coordenadas espaciais dos pontos a auscultar

Tendo os pontos 2D de auscultação determinados, passou-se ao cálculo das coordenadas espaciais destes pontos. Este cálculo foi realizado através do mesmo algoritmo mencionado anteriormente. Ou seja, foi utilizada a função *calc_spatials()* onde os seus parâmetros de entrada foram a imagem de profundidade, e as duas coordenadas x e y de cada ponto de auscultação 2D. Esta função devolve a localização espacial 3D de cada ponto 2D, calculado anteriormente. A Figura 5.27 mostra um *frame* das coordenadas espaciais dos pontos a auscultar.

Para obter valores mais precisos, durante 250 instantes em que há valores disponíveis de coordenadas espaciais para cada ponto, o algoritmo acumula os seus valores nas mesmas variáveis e no fim faz uma média. Estes valores finais são também convertidos para metros.

5.3.5 Realização da auscultação

Obtidas as coordenadas espaciais dos pontos a auscultar, o braço robótico foi movimentado para as mesmas posições de junta mencionadas anteriormente quando foi calculado o *offset* e também com a mesma velocidade e aceleração desse movimento.

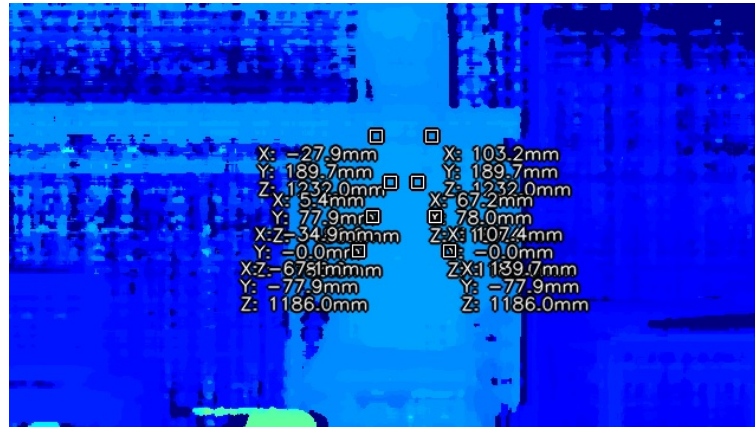


Figura 5.27: Coordenadas espaciais 3D dos pontos a auscultar

Seguidamente, foi calculada a distância entre os pontos de auscultação para o braço robótico realizar uma auscultação contínua, sem ter que voltar à posição inicial. Este cálculo começou no ponto de auscultação *L2*, porque o braço robótico UR3e não tem alcance suficiente para chegar aos pontos *L1* e *R1*.

Para calcular estas distâncias, primeiramente, foi feita a média entre o valor da coordenada *y* dos dois pontos de cada nível, ou seja:

$$y1 = (eL1_y + eR1_y)/2$$

$$y2 = (eL2_y + eR2_y)/2$$

$$y3 = (eL3_y + eR3_y)/2$$

$$y4 = (eL4_y + eR4_y)/2$$

Esta média foi calculada porque os pontos de auscultação no lado direito e no lado esquerdo devem estar na mesma coordenada *y*, para auscultar sempre no mesmo sítio, mas no lado oposto do tronco.

De seguida, as distâncias são calculadas da seguinte forma:

$$dist_1x = eL2_x - robot_offset_x \quad dist_1y = y2 - robot_offset_y$$

$$dist_2x = eR2_x - eL2_x \quad dist_2y = 0$$

$$dist_3x = eR3_x - eR2_x \quad dist_3y = y3 - y2$$

$$dist_4x = eL3_x - eR3_x \quad dist_4y = 0$$

$$dist_5x = eL4_x - eL3_x \quad dist_5y = y4 - y3$$

$$dist_6x = eR4_x - eL4_x \quad dist_6y = 0$$

$$dist_7x = -eR4_x \quad dist_7y = -y4$$

onde, *robot_offset_x* e *robot_offset_y* correspondem ao valor das coordenadas *x* e *y*, respetivamente, do valor do *offset* calculado anteriormente.

As variáveis que contêm um *e* no início correspondem à coordenada espacial do respectivo ponto. A última distância corresponde a colocar o robô de volta na posição padrão. A Figura 5.28 mostra o trajeto que o braço robótico vai percorrer.

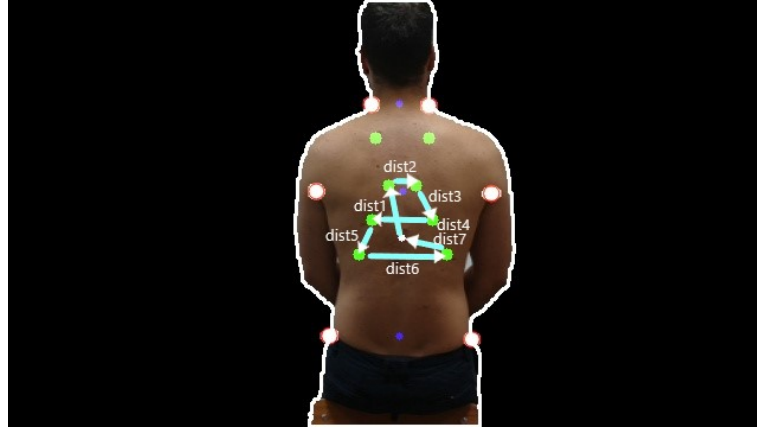


Figura 5.28: Trajeto realizado pelo braço robótico

Para movimentar o braço robótico tendo em conta estas distâncias, foi utilizada a função *movel()* [62]. Com esta função é possível dar ordem ao braço para se movimentar uma certa distância, tendo em conta a posição atual dele. Estas movimentações do braço robótico entre pontos foram realizadas a uma velocidade de 0,2 m/s e a uma aceleração de 0,2 m/s².

Quando o robô chegava ao ponto de auscultação, este avançava em direção às costas da pessoa, a uma velocidade de 0,2 m/s e aceleração de 0,2 m/s², enquanto que a força que ele aplicava na ponta do *gripper* era monitorizada ao longo do eixo *x* representado na Figura 5.29. Sempre que esta força ultrapassava a força recebida inicialmente (quando o braço robótico chegava ao ponto de auscultação) em 3 N, este parava 10 seg (simula a auscultação). De seguida, o braço robótico afasta-se das costas da pessoa, e depois avançava para o próximo ponto.

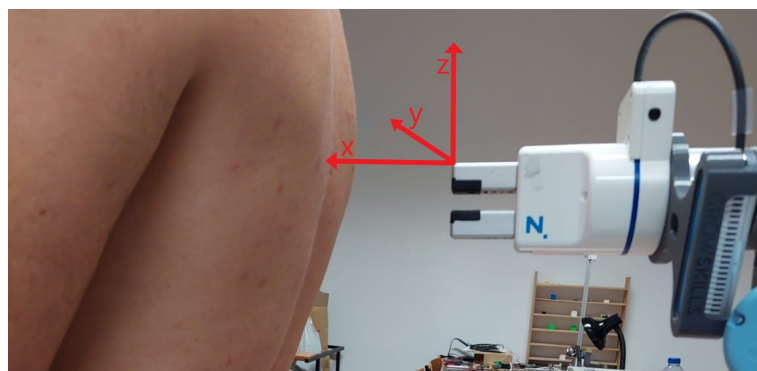


Figura 5.29: Referencial do braço robótico

Para obter a força aplicada na ponta do *gripper* foi utilizada a interface RTDE. Esta interface fornece uma maneira de recolher informação do braço robótico, permitindo sincronizar variáveis que podem ser escolhidas pelo cliente, para serem escritas e lidas. Estas devem, por sua vez, estar contidas nos pacotes reais de sincronização - receita. Cada receita tem um *id* exclusivo. O RTDE Control Package Setup Outputs devolve os valores das variáveis, na mesma ordem com que foram pedidas. O RTDE foi inicializado na porta 30004, como já foi dito anteriormente, o *host* foi o IP do braço robótico e com uma frequência de 125 Hz. Mais detalhe encontra-se presente no Código C.4 disponível no Anexo C.

A Figura 5.30 apresenta o fluxograma deste algoritmo de forma mais sucinta e direta.

5.4 Conclusão do capítulo 5

Neste capítulo foram apresentadas as soluções para concluir os objetivos propostos. Aqui é explicado como foi feita a junção entre os conceitos teóricos, o *hardware* e as interfaces, descritos anteriormente, de modo a desenvolver uma solução.

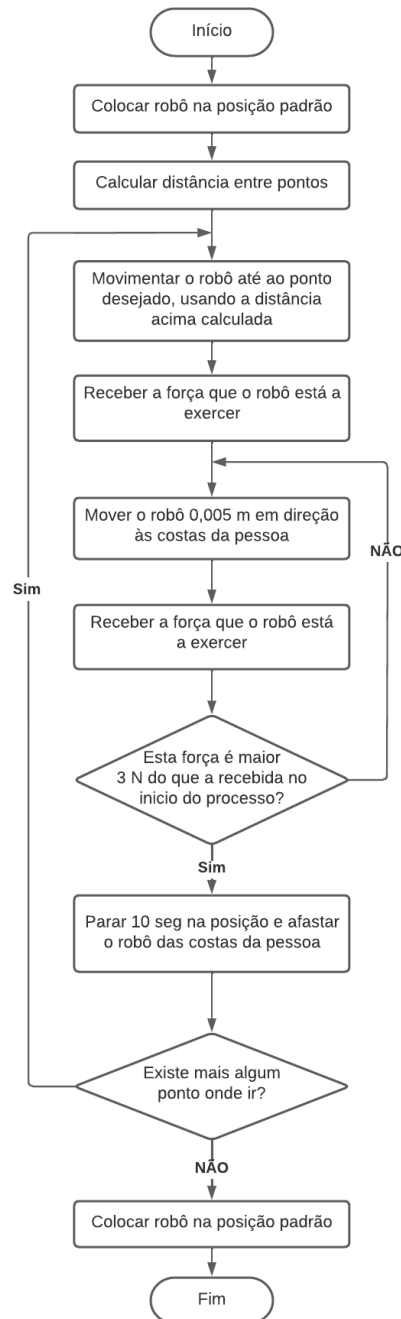


Figura 5.30: Algoritmo de controlo do braço robótico

Capítulo 6

Resultados

Neste capítulo são analisados os resultados da solução proposta.

Para testar a plataforma desenvolvida para auscultação autónoma foi realizada uma simulação de auscultação, ou seja, o *gripper* apenas encosta nas costas do paciente. O paciente, durante todo este processo, está em pé encostado às costas de uma cadeira para obter uma melhor estabilidade. Estas simulações foram realizadas em quatro pacientes presentes e dois não presentes no *Dataset* utilizado para treinar o modelo de previsão dos pontos de referência.

6.1 Pacientes presentes no *Dataset*

Primeiramente, realizou-se o cálculo do *offset* do alinhamento da câmara com o braço robótico, demonstrado nas Figuras 5.19 e 5.20. Este cálculo foi realizado com sucesso, sendo obtidos os valores 2D do *offset* e, de seguida, as coordenadas espaciais 3D mostradas na Figura 5.20. Este *offset* é uma questão de poucos milímetros, mas faz diferença na colocação do *gripper* nas costas da pessoa no sítio correto.

De seguida, foram obtidas as *frames* das costas dos quatro pacientes usados neste teste (ver Figura 6.1), e as suas respetivas extrações do fundo de imagem (ver Figura 6.2) e foram previstos os pontos de referência através do modelo de previsão treinado, demonstrados na Figura 6.3.



Figura 6.1: *Frames* das costas dos pacientes utilizadas para a previsão dos pontos de referência



Figura 6.2: Extração do fundo das *frames* das costas dos pacientes



Figura 6.3: Pontos de referência previstos nas costas dos quatro pacientes

Por conseguinte, foram calculados os pontos de auscultação 2D, através dos pontos de referência. Estes pontos de auscultação, demonstrados a verde na Figura 6.4, foram calculados com sucesso, visto que os pontos de referência também foram previstos com sucesso. As respetivas coordenadas espaciais 3D dos pontos de auscultação também foram calculadas, como é possível ver na Figura 6.5.

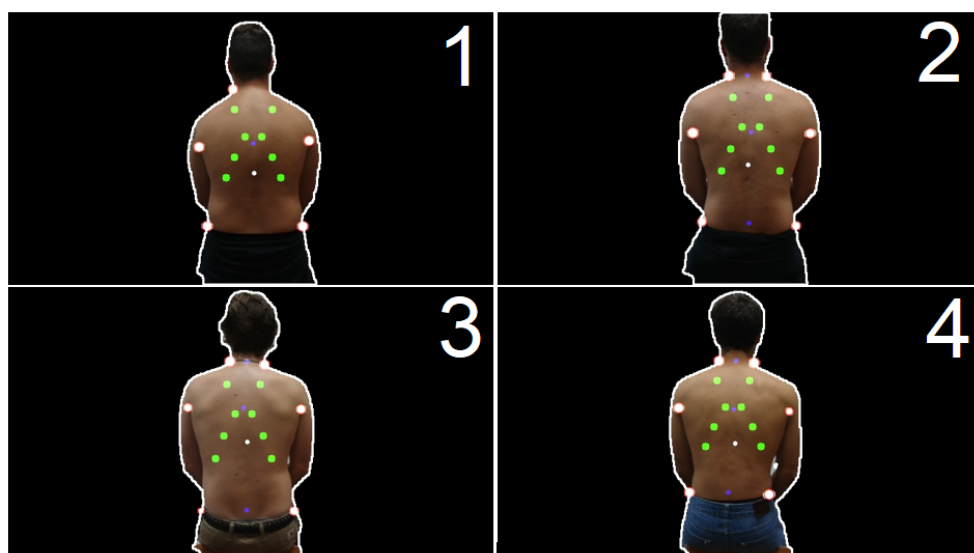


Figura 6.4: Pontos de auscultação calculados nos quatro pacientes

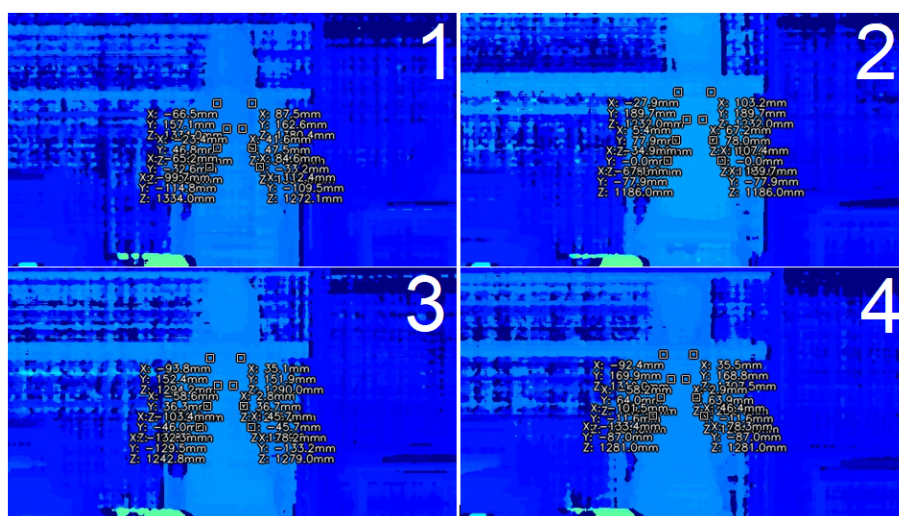


Figura 6.5: Coordenadas espaciais 3D dos pontos de auscultação dos quatro pacientes

O *gripper* do braço robótico foi colocado nos pontos de auscultação calculados nos quatro pacientes, como se pode ver pelas Figuras 6.6, 6.7, 6.8 e 6.9.

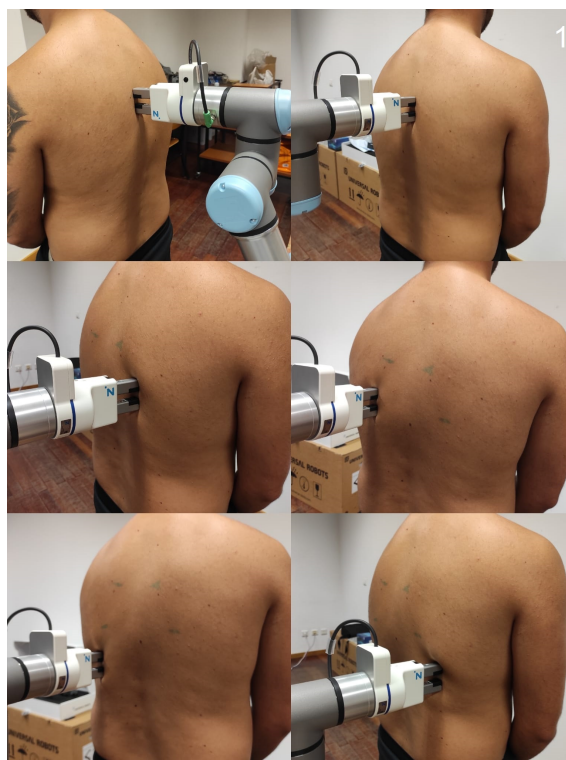


Figura 6.6: Braço robótico nos pontos de auscultação do paciente 1

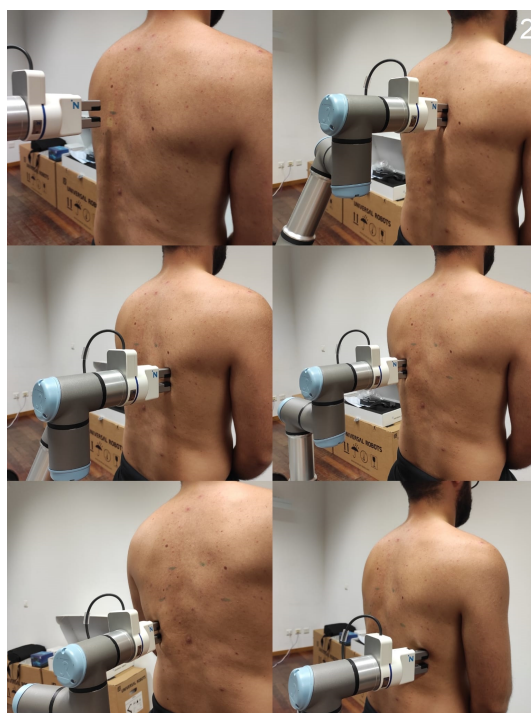


Figura 6.7: Braço robótico nos pontos de auscultação do paciente 2



Figura 6.8: Braço robótico nos pontos de auscultação do paciente 3

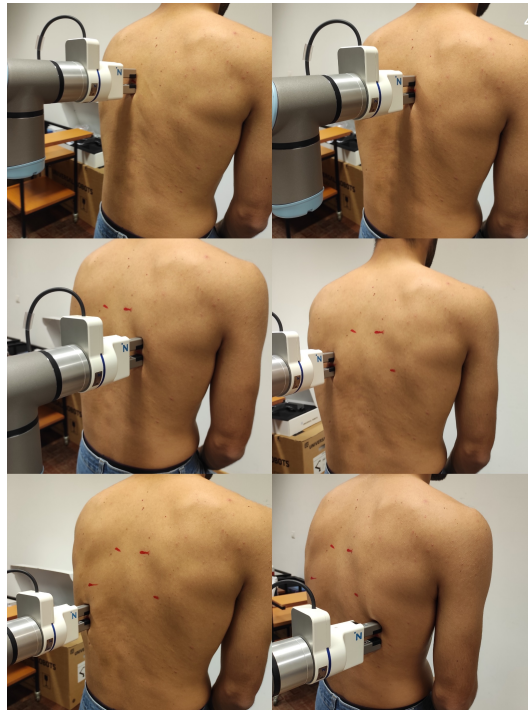


Figura 6.9: Braço robótico nos pontos de auscultação do paciente 4

O braço robótico foi com sucesso aos pontos de auscultação desejados nos pacientes 1, 2 e 4, exceto aos pontos *L1* e *R1*, uma vez que não foram introduzidos no cálculo das distâncias, porque o robô não tem alcance suficiente para lá chegar. No paciente 3, o braço robótico não conseguiu ir aos pontos *L4* e *R4*, porque ao descer do ponto *L3* para o ponto *L4*, a junta *elbow* bateu na mesa de suporte do braço robótico. Isto deveu-se ao facto deste paciente ser mais pequeno, porque pelo cálculo das coordenadas espaciais 3D, como está demonstrado na Figura 6.5, o paciente 3 possui o valor da coordenada *y* nos pontos *L4* e *R4* mais baixo entre os quatro pacientes. Os pontos auscultados nos quatro pacientes estão ilustrados a marcador na Figura 6.10.

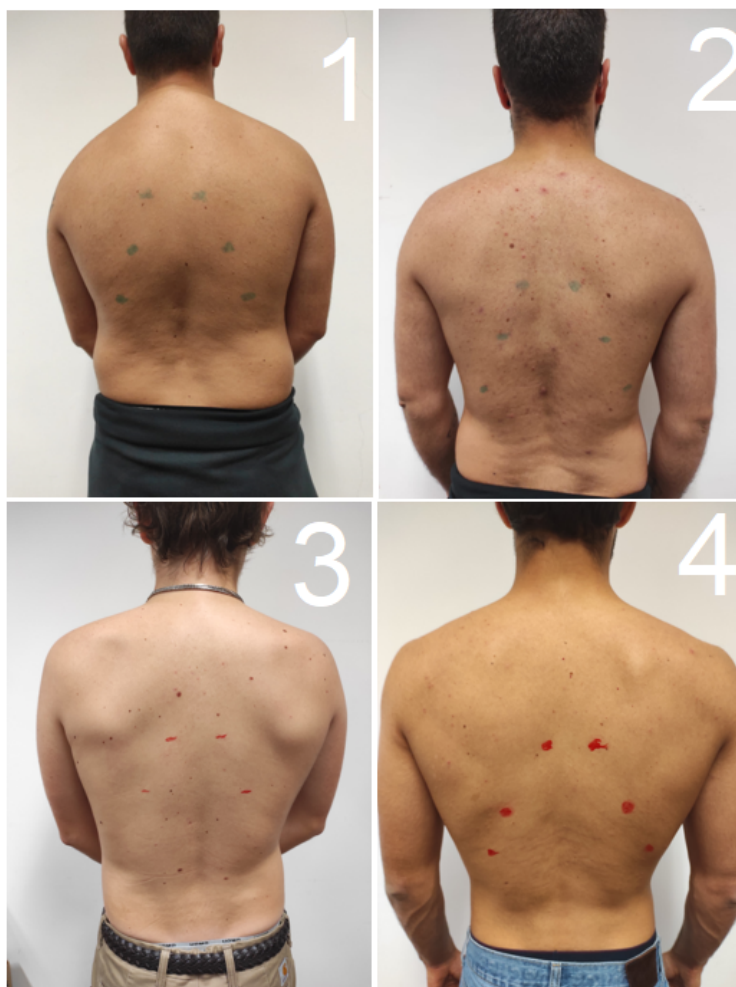


Figura 6.10: Pontos auscultados nos quatro pacientes

Como é possível ver pela Figura 6.10, os pontos auscultados pelo braço robótico são semelhantes aos pontos 2D calculados anteriormente pelo algoritmo. Estes podem ter algumas divergências devido ao paciente não conseguir estar 100% imóvel durante todo o processo de auscultação.

De seguida, o sistema vai ser testado na auscultação de dois pacientes que não estiveram presentes na criação do *Dataset*. A obtenção dos seguintes resultados tem como objetivo verificar se o modelo treinado é robusto o suficiente, ou se apenas deteta os pontos de referência em costas de pacientes que estiveram presentes na criação do *Dataset*.

6.2 Pacientes não presentes no *Dataset*

O processo realizado com os dois pacientes não presentes no *Dataset* é igual ao anteriormente descrito. Sendo, primeiramente, obtidas as *frames* das costas dos pacien-

tes, presentes na Figura 6.11. De seguida, é realizada a extração do fundo da imagem, demonstrada na Figura 6.12 e, por último, foram previstos com sucesso os pontos de referência, ilustrados na Figura 6.13.



Figura 6.11: *Frames* das costas dos pacientes não presentes no *Dataset* utilizadas para a previsão dos pontos de referência



Figura 6.12: Extração do fundo das *frames* das costas dos pacientes não presentes no *Dataset*

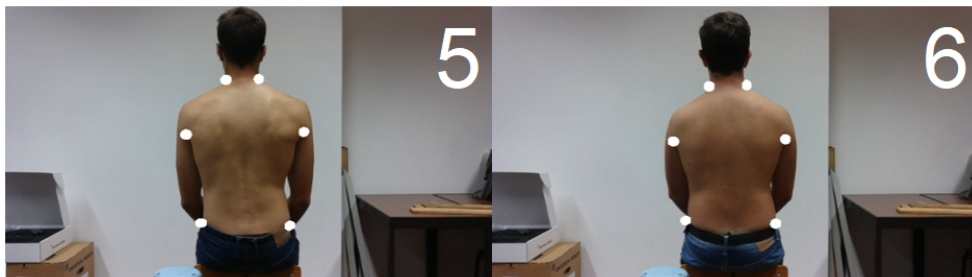


Figura 6.13: Pontos de referência previstos nas costas dos dois pacientes

Em consequência, são calculados os pontos de auscultação 2D, através dos pontos de referência, mostrados a verde na Figura 6.14. As respetivas coordenadas espaciais 3D dos pontos de auscultação também foram calculadas com sucesso, como é possível ver na Figura 6.15.

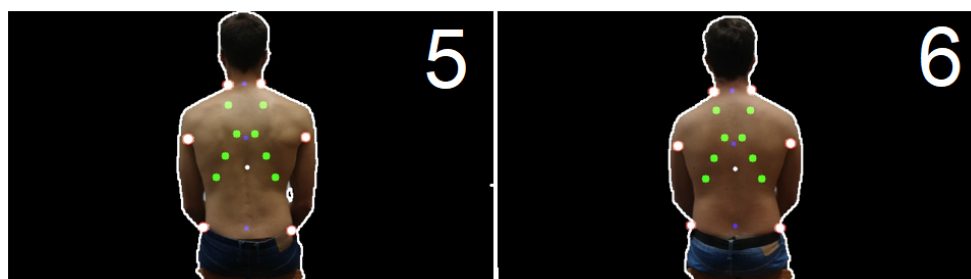


Figura 6.14: Pontos de auscultação calculados nos dois pacientes

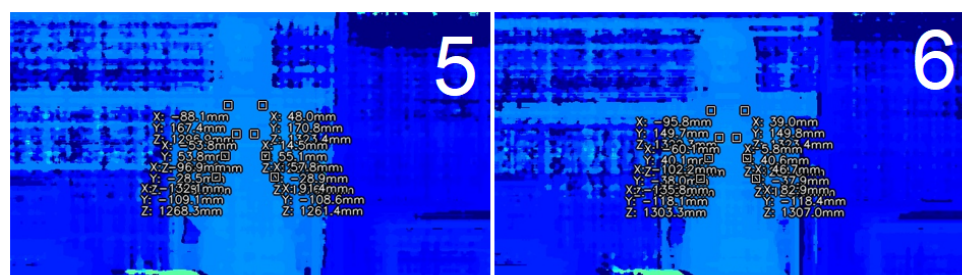


Figura 6.15: Coordenadas espaciais 3D dos pontos de auscultação dos dois pacientes

Por fim, o *gripper* do braço robótico foi colocado nos pontos de auscultação calculados nestes dois pacientes, como se pode ver pelas Figuras 6.16 e 6.17.

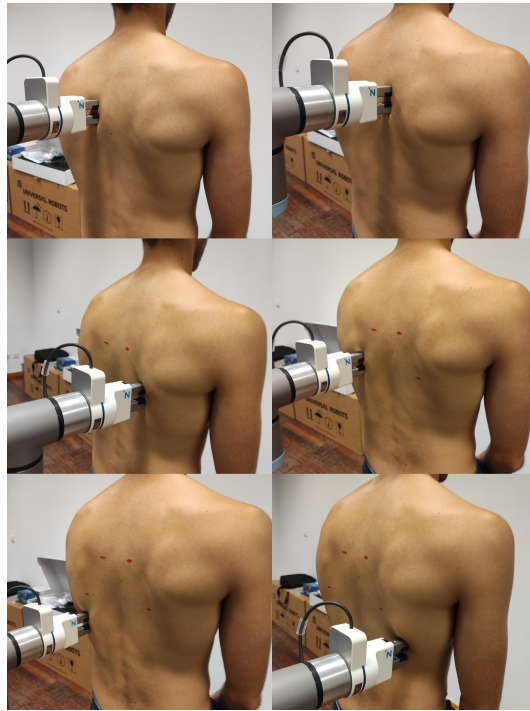


Figura 6.16: Braço robótico nos pontos de auscultação do paciente 5

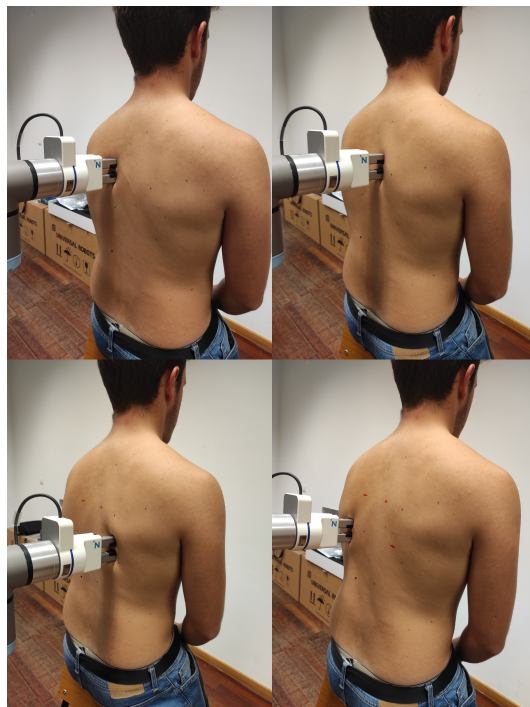


Figura 6.17: Braço robótico nos pontos de auscultação do paciente 6

O braço robótico, tal como nos testes anteriormente realizados, não foi aos pontos *L1* e *R1*. Este, no paciente 5, conseguiu alcançar os 6 pontos de auscultação desejados com sucesso, mas no paciente 6, tal como aconteceu com o paciente 3, quando o braço robótico tentou alcançar os pontos *L4* e *R4*, a junta *elbow* bateu na mesa de suporte do braço robótico, como é possível ver na Figura 6.18. Ao analisar a Figura 6.15, é possível verificar que o paciente 6 tem a segunda coordenada *y*, dos pontos *L4* e *R4*, mais baixa de todos. A partir das imagens de profundidade (Figura 6.5 e 6.15) é possível assumir que o braço robótico consegue alcançar todos os pontos de auscultação desejados com sucesso, em todos os pacientes que tenham uma coordenada *y*, nos pontos *L4* e *R4*, acima de -114.0 mm.



Figura 6.18: Colisão da junta *elbow* do braço robótico com a bancada onde se encontra instalado, ao tentar alcançar o ponto *L4* do paciente 6

Na Figura 6.19 estão representados a marcador os pontos em que o *gripper* do braço robótico tocou nas costas dos pacientes. Comparando estes pontos a marcador com os pontos 2D calculados anteriormente é possível verificar que são muito semelhantes. Estes, tais como os pontos a marcador obtidos nos pacientes presentes no *Dataset*, apresentam apenas algumas discrepâncias devido ao movimento do paciente.



Figura 6.19: Pontos auscultados nos dois pacientes não presentes no *Dataset*

6.3 Conclusão do capítulo 6

Neste capítulo foram apresentados os resultados da simulação de auscultação, utilizando a plataforma desenvolvida. Esta plataforma foi testada em seis pacientes, sendo que quatro destes pacientes estavam presentes no *Dataset* criado. Através destes testes foi possível concluir que o modelo treinado através do *Dataset* é robusto o suficiente para prever os pontos de referência em diferentes pacientes. O braço robótico consegue alcançar os pontos de auscultação desejados com sucesso, dependendo da estrutura física do paciente, nomeadamente o seu comprimento de costas.

Capítulo 7

Conclusão

Neste capítulo serão expostas as conclusões gerais e o trabalho futuro deste projeto.

O trabalho realizado nesta dissertação apresenta um modelo conceptual de uma plataforma de auscultação robótica que permite simular o posicionamento autónomo do estetoscópio para satisfazer a segurança do paciente em termos de força de contacto. Os resultados preliminares demonstraram que a plataforma robótica permite estimar as posições de auscultação para ouvir os sons de quatro válvulas cardíacas, com base nas informações de referência da superfície do corpo, e simula a colocação de um estetoscópio mantendo a força de contacto. A plataforma robótica desenvolvida tem o potencial de abordar a questão crítica do aumento do número de pacientes atendidos por profissional de saúde. A utilização desta tecnologia pode aumentar ainda mais a eficiência do rastreio de sinais clínicos anormais, incluindo a COVID-19.

Como trabalho futuro, esta plataforma deverá sofrer algumas alterações. Para uma melhor precisão na colocação do estetoscópio no ponto de auscultação, o paciente deve estar mais estável, ou seja, este deverá estar sentado, uma vez que esta também é a melhor posição para obter melhores resultados na auscultação. O braço robótico também terá que ser alterado para um com um alcance superior, uma vez que o utilizado tinha um alcance muito reduzido, limitando os seus movimentos, e para ter uma maior zona de trabalho este não deve ser montado numa mesa. A câmara também deverá ser colocada numa posição fixa, onde esta não seja movimentada com qualquer mínimo toque. Para concluir a finalidade desta plataforma, deve ser introduzido um estetoscópio digital no *gripper*, ou criar uma ferramenta que tenha um estetoscópio digital incluído para substituir pelo *gripper*.

Referências Bibliográficas

- [1] “automation | technology, types, rise, history, examples.” <https://www.britanica.com/technology/automation>. Accessed: 2022-07-30.
- [2] M. R. Montinari and S. Minelli, “The first 200 years of cardiac auscultation and future perspectives,” *Journal of Multidisciplinary Healthcare*, vol. 12, p. 183, 2019. doi: 10.2147/JMDH.S193904.
- [3] “Dia mundial do pulmão: doenças pulmonares são a terceira causa de morte a nível mundial – raios – o seu jornal de saúde diário.” <https://raiox.pt/dia-mundial-do-pulmao-doencas-pulmonares-sao-a-terceira-caoa-de-morte-a-nivel-mundial/>. Accessed: 2022-07-30.
- [4] S. Leng, R. S. Tan, K. T. C. Chai, C. Wang, D. Ghista, and L. Zhong, “The electronic stethoscope,” *BioMedical Engineering OnLine*, vol. 14, 7 2015. doi: 10.1186/S12938-015-0056-Y.
- [5] “Investimento na saúde aumentou, mas dados revelam falta de profissionais.” <https://expresso.pt/sociedade/2021-09-30-Investimento-na-saude-aumentou-mas-dados-revelam-falta-de-profissionais-3365411e>. Accessed: 2022-07-30.
- [6] Y. Ma, X. Xu, Q. Yu, Y. Zhang, Y. Li, J. Zhao, and G. Wang, “Lungbrn: A smart digital stethoscope for detecting respiratory disease using bi-resnet deep learning algorithm,” *BioCAS 2019 - Biomedical Circuits and Systems Conference, Proceedings*, 10 2019. doi: 10.1109/BIOCAS.2019.8919021.
- [7] “Auscultation: Definition, purpose & procedure.” <https://my.clevelandclinic.org/health/diagnostics/23080-auscultation>. Accessed: 2022-07-30.
- [8] G. Altan, Y. Kutlu, and A. Gökçen, “Chronic obstructive pulmonary disease severity analysis using deep learning on multi-channel lung sounds,” *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 28, pp. 2979–2996, 2020. doi: 10.3906/ELK-2004-68.

- [9] “Tytohome™ telehealth equipment - tytocare for professionals.” <https://www.tytocare.com/products/tytohome/>. Accessed: 2022-07-30.
- [10] “Estetoscópio core digital.” https://www.3m.com.pt/3M/pt_PT/Littmann-P/T/advantages/core-digital-stethoscope/. Accessed: 2022-07-30.
- [11] “3m™ littmann® core digital stethoscope, 8480, black chestpiece, tube, stem and headset, 27 inch.” https://www.3m.com/3M/en_US/p/d/v101191252/. Accessed: 2022-07-30.
- [12] “Best electronic stethoscopes (2022 updated reviews).” <https://www.stethoscopesguides.com/best-electronic-stethoscopes/>. Accessed: 2022-07-30.
- [13] “Thinklabs one - digital stethoscope.” <https://www.thinklabs.com/>. Accessed: 2022-07-30.
- [14] S. H. Lee, Y. S. Kim, M. K. Yeo, M. Mahmood, N. Zavanelli, C. Chung, J. Y. Heo, Y. Kim, S. S. Jung, and W. H. Yeo, “Fully portable continuous real-time auscultation with a soft wearable stethoscope designed for automated disease diagnosis,” *Science advances*, vol. 8, p. eabo5867, 5 2022. doi: 10.1126/SCIADV.AB05867/SUPPL_FILE/SCIADV.AB05867_MOVIES_S1_TO_S10.ZIP.
- [15] R. Tsumura, Y. Koseki, N. Nitta, and K. Yoshinaka, “Toward fully automated robotic platform for remote auscultation,” 2022. doi: 10.48550/ARXIV.2201.05272.
- [16] Y. Zhu, A. Smith, and K. Hauser, “Automated heart and lung auscultation in robotic physical examinations,” 2022. doi: 10.48550/ARXIV.2201.09511.
- [17] “Machine learning, explained.” <https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained>. Accessed: 2022-07-30.
- [18] “Understanding semantic segmentation with unet.” <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>. Accessed: 2022-07-30.
- [19] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015. doi: 10.48550/ARXIV.1505.04597.
- [20] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” 2014. doi: 10.48550/ARXIV.1411.4038.
- [21] “What is unet?.” <https://medium.com/analytics-vidhya/what-is-unet-157314c87634>. Accessed: 2022-07-30.
- [22] “How u-net works?.” <https://developers.arcgis.com/python/guide/how-unet-works/>. Accessed: 2022-07-30.

- [23] “Mobilenet convolutional neural network machine learning algorithms.” <https://medium.com/analytics-vidhya/image-classification-with-mobile-net-cc6fbb2cd470>. Accessed: 2022-07-30.
- [24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” 2017. doi: 10.48550/ARXIV.1704.04861.
- [25] “Google ai blog: Mobilenetv2: The next generation of on-device computer vision networks.” <https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html>. Accessed: 2022-07-30.
- [26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2018. doi: 10.48550/ARXIV.1801.04381.
- [27] J. Jing, Z. Wang, M. Rättsch, and H. Zhang, “Mobile-unet: An efficient convolutional neural network for fabric defect detection,” *Textile Research Journal*, vol. 92, 2020. doi: 10.1177/0040517520928604.
- [28] “Unet segmentation with pretrained mobilenetv2 as encoder.” <https://idiotdeveloper.com/unet-segmentation-with-pretrained-mobilenetv2-as-encoder/>. Accessed: 2022-07-30.
- [29] “A simple guide to the versions of the inception network.” <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>. Accessed: 2022-07-30.
- [30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014. doi: 10.48550/ARXIV.1409.4842.
- [31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” pp. 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.
- [32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *AAAI Conference on Artificial Intelligence*, vol. 31, 02 2016. doi: 10.1609/aaai.v31i1.11231.
- [33] “Keras applications.” <https://keras.io/api/applications/>. Accessed: 2022-07-30.
- [34] “Inceptionresnetv2 simple introduction.” <https://medium.com/@zahraelhamraoui1997/inceptionresnetv2-simple-introduction-9a2000edcbb6>. Accessed: 2022-07-30.

- [35] “What is data augmentation? techniques & examples in 2022.” <https://research.aimultiple.com/data-augmentation/>. Accessed: 2022-07-30.
- [36] “Intuition of adam optimizer.” <https://www.geeksforgeeks.org/intuition-of-adam-optimizer/>. Accessed: 2022-07-30.
- [37] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. doi: 10.48550/ARXIV.1412.6980.
- [38] “Adam.” <https://keras.io/api/optimizers/adam/>. Accessed: 2022-07-30.
- [39] “Optimization techniques for gradient descent.” <https://www.geeksforgeeks.org/optimization-techniques-for-gradient-descent/>. Accessed: 2022-07-30.
- [40] “Understanding dice loss for crisp boundary detection.” <https://medium.com/ai-salon/understanding-dice-loss-for-crisp-boundary-detection-bb30c2e5f62b>. Accessed: 2022-07-30.
- [41] “Common classification model evaluation metrics.” <https://towardsdatascience.com/common-classification-model-evaluation-metrics-2ba0a7a7436e>. Accessed: 2022-07-30.
- [42] “Evaluating image segmentation models.” <https://www.jeremyjordan.me/evaluating-image-segmentation-models/>. Accessed: 2022-07-30.
- [43] “Precision vs recall | precision and recall machine learning.” <https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/>. Accessed: 2022-07-30.
- [44] “Ur3e collaborative robot arm that automates almost anything.” <https://www.universal-robots.com/products/ur3-robot/>. Accessed: 2022-07-30.
- [45] “Ur3e technical specifications.” <https://www.universal-robots.com/media/1807464/ur3e-rgb-fact-sheet-landscape-a4.pdf>. Accessed: 2022-07-30.
- [46] “Oak-d — depthai hardware documentation 1.0.0 documentation.” <https://docs.luxonis.com/projects/hardware/en/latest/pages/BW10980AK.html>. Accessed: 2022-07-30.
- [47] “Introduction to oak-d and depthai.” <https://learnopencv.com/introduction-to-opencv-ai-kit-and-depthai/>. Accessed: 2022-07-30.
- [48] “Remote control via tcp/ip - 16496.” <https://www.universal-robots.com/articles/ur/interface-communication/remote-control-via-tcpip/>. Accessed: 2022-07-30.

- [49] “Overview of client interfaces - 21744.” <https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/>. Accessed: 2022-07-30.
- [50] “Universal robots - offline simulator - cb-series - non linux - ursim 3.14.3.” <https://www.universal-robots.com/download/software-cb-series/simulator-non-linux/offline-simulator-cb-series-non-linux-ursim-3143/>. Accessed: 2022-07-30.
- [51] “What is tensorflow? the machine learning library explained.” <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>. Accessed: 2022-07-30.
- [52] “Pytorch.” <https://pytorch.org/>. Accessed: 2022-09-24.
- [53] “Apache mxnet | a flexible and efficient library for deep learning..” <https://mxnet.apache.org/versions/1.9.1/>. Accessed: 2022-09-24.
- [54] “About keras.” <https://keras.io/about/>. Accessed: 2022-07-30.
- [55] “Depthai api documentation — depthai documentation.” <https://docs.luxonis.com/projects/api/en/latest/>. Accessed: 2022-07-30.
- [56] “Depthai pipeline overview: Creating a complex pipeline.” <https://learnopencv.com/depthai-pipeline-overview-creating-a-complex-pipeline/>. Accessed: 2022-07-30.
- [57] “Socket.” <https://pt.slideshare.net/Baguiasri/socket-30350413>. Accessed: 2022-07-30.
- [58] “Tcp server-client implementation in c.” <https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/>. Accessed: 2022-07-30.
- [59] “Real-time data exchange (rtde) guide - 22229.” <https://www.universal-robots.com/articles/ur/interface-communication/real-time-data-exchange-rtde-guide/>. Accessed: 2022-07-30.
- [60] “Github - depthai-experiments/gen2-calc-spatials-on-host at master · luxonis/depthai-experiments.” <https://github.com/luxonis/depthai-experiments/tree/master/gen2-calc-spatials-on-host#calculate-spatial-coordinates-on-the-host>. Accessed: 2022-07-30.
- [61] “Github - nikhilroxtomar/human-image-segmentation-with-deeplabv3plus-in-tensorflow: Deeplabv3+ with squeeze and excitation network for human image segmentation in tensorflow 2.5.0.” <https://github.com/nikhilroxtomar/Human-Image-Segmentation-with-DeepLabV3Plus-in-TensorFlow>. Accessed: 2022-07-30.

- [62] U. Robots, “The urscript programming language for e-series.” https://s3-eu-west-1.amazonaws.com/ur-support-site/105198/scriptManual_SW5.10.pdf. Accessed: 2022-09-24.
- [63] J. Jing, Z. Wang, M. Rättsch, and H. Zhang, “Mobile-unet: An efficient convolutional neural network for fabric defect detection,” *Textile Research Journal*, vol. 92, 2020. doi: 10.1177/0040517520928604.
- [64] P.-L. Pröve, “Mobilenetv2: Inverted residuals and linear bottlenecks.” <https://towardsdatascience.com/mobilenetv2-inverted-residuals-and-linear-bottlenecks-8a4362f4ffd5>.

Anexo A

MobileNetV1

A.1 Convolução com profundidade

As convoluções profundas são utilizadas para aplicar um único filtro em cada canal de entrada, ao contrário de uma convolução normal na qual os filtros são aplicados a todos os canais de entrada [24, 63].

Tomando em consideração uma convolução normal, descrita na Figura A.1.

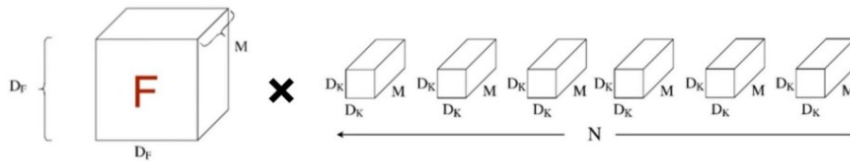


Figura A.1: Convolução normal [63]

A partir da imagem acima, o custo computacional pode ser calculado a partir da equação A.1 [24, 63]:

$$D_K^2 * M * N * D_F^2 \quad (\text{A.1})$$

Onde D_K o tamanho do núcleo de convolução e D_F representa as dimensões espaciais do mapa de características de entrada. M e N são o número de canais de entrada e saída, respectivamente [24, 63].

Para uma convolução normal, o custo computacional depende multiplicativamente do número de canais de entrada e de saída e das dimensões espaciais do mapa de características de entrada e do núcleo de convolução [24, 63].

O caso da convolução profunda, como é possível ver na Figura A.2, contém um mapa da característica de entrada da dimensão $D_F * D_F$ e um número M de *Kernels* do tamanho do canal 1 [24, 63].

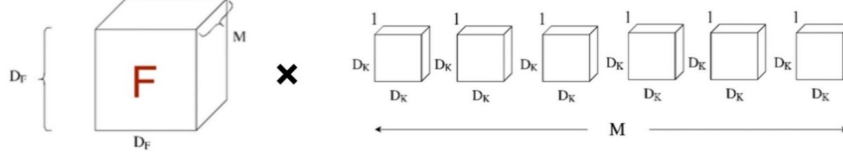


Figura A.2: Convolução profunda [63]

De acordo com a imagem acima, é possível ver claramente que o custo computacional total pode ser calculado através da equação A.2 [24, 63]:

$$D_K^2 * M * D_F^2 \quad (\text{A.2})$$

Contudo, este método só é utilizado para filtrar o canal de entrada.

A.2 Convolução pontual

Uma vez que a convolução em profundidade é apenas utilizada para filtrar o canal de entrada, esta não os combina para produzir novas características. Consequentemente, é feita uma camada adicional chamada camada de convolução pontual, descrita na Figura A.3, que calcula uma combinação linear da saída da convolução de profundidade utilizando uma convolução de 1×1 [24, 63].

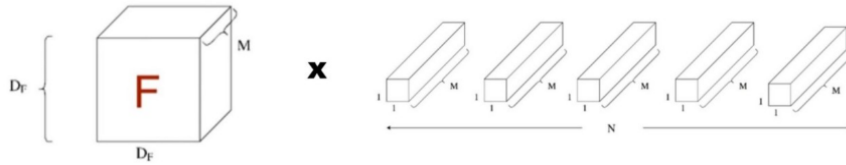


Figura A.3: Convolução pontual [63]

Conforme a imagem, o custo computacional pode ser calculado através da equação A.3 [24, 63]:

$$M * N * D_F^2 \quad (\text{A.3})$$

Assim, o custo computacional total das convoluções profundas separadas pode ser calculado através da equação A.4 [24, 63]:

$$D_K^2 * M * D_F^2 + M * N * D_F^2 \quad (\text{A.4})$$

Comparando-o com o custo computacional da convolução normal, é obtida uma redução no cálculo, que pode ser expressa na equação A.5:

$$\frac{D_K^2 * M * D_F^2 + M * N * D_F^2}{D_K^2 * M * N * D_F^2} = \frac{1}{N} + \frac{1}{D_K^2} \quad (\text{A.5})$$

Através do exemplo dado abaixo, é possível verificar a eficácia desta convolução separável em profundidade.

Utilizando $N = 1024$ e $D_k = 3$ na equação, é obtido 0,112, ou, noutras palavras, a convolução normal tem 9 vezes mais o número de multiplicações do que a convolução profunda [63].

A.3 Arquitetura MobileNetV1

A partir da Figura A.4, é possível ver que cada camada de convolução é seguida de uma *batch normalization* e um ReLU. Além disso, foi introduzido um *pooling* final médio imediatamente antes da camada totalmente ligada para reduzir a dimensão espacial a 1 [24, 63], como é possível ver na arquitetura da MoblieNetV1 presente na Figura A.6.

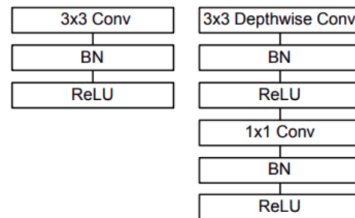


Figura A.4: Na esquerda, camada convolucional normal com *batch normalization* e ReLU. Na direita, convolução profunda separável com camadas em profundidade e em pontos, seguida de *batch normalization* e ReLU [24, 63]

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$	Conv dw / s1 $3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	Conv / s1 $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figura A.5: Arquitetura MobileNetV1 [24, 63]

Anexo B

MobileNetV2

B.1 Resíduos invertidos

Os blocos residuais, mostrados na Figura B.1, ligam o início e o fim de um bloco convolutivo com uma *skip connection*. Ao adicionar estes dois estados, a rede tem a oportunidade de aceder a ativações anteriores que não foram modificadas no bloco convolutivo. Esta abordagem revelou-se essencial para a construção de redes com grande profundidade [26, 64].

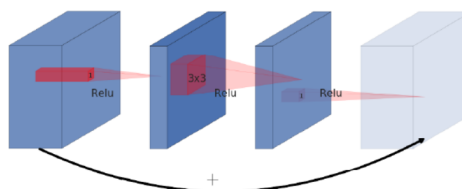


Figura B.1: Um bloco residual liga largas camadas com uma *skip connection* enquanto as camadas intermédias são estreitas [26, 64]

Olhando um pouco mais de perto para a *skip connection*, nota-se que um bloco residual original segue uma abordagem ampla->estreita->ampla relativamente ao número de canais. A entrada tem um elevado número de canais, que são comprimidos com uma convolução 1x1 menos dispendiosa. Desta forma, a seguinte convolução 3x3 tem muito menos parâmetros. Com o propósito de adicionar uma entrada e uma saída no final, o número de canais é novamente aumentado utilizando outra convolução 1x1 [26, 64].

Por outro lado, a MobileNetV2 segue uma abordagem contrária à acima descrita, ou seja, estreita->larga->estreita. Neste caso esta primeiramente alarga a rede usando uma convolução 1x1 porque a seguinte convolução 3x3 em profundidade já reduz significati-

vamente o número de parâmetros. Depois, realiza outra convolução 1×1 para reduzir a rede de modo a corresponder ao número inicial de canais [26, 64], como é possível ver pela Figura B.2.

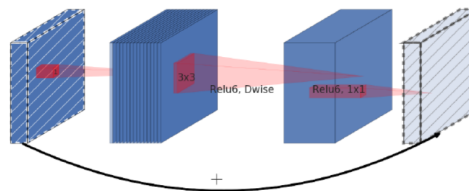


Figura B.2: Um bloco residual invertido liga camadas estreitas com uma *skip connection* enquanto as camadas intermédias são largas [26, 64]

Os autores descrevem esta ideia como um bloco residual invertido porque existem *skip connections* entre partes estreitas da rede, o que é o oposto de como funciona uma ligação residual original [26, 64].

B.2 Engarrafamentos lineares

A razão pela qual são utilizadas funções de ativação não lineares em redes neuronais é pelo facto de as múltiplas multiplicações de matrizes não poderem ser reduzidas a uma única operação numérica. Deste modo, é possível construir redes neuronais que têm múltiplas camadas. Ao mesmo tempo, a função de ativação ReLU, que é normalmente utilizada em redes neuronais, descarta valores inferiores a 0. Esta perda de informação pode ser combatida aumentando o número de canais de modo a aumentar a capacidade da rede [26].

Com os blocos residuais invertidos, é feito o oposto e são reduzidas as camadas onde as *skip connections* estão ligadas. Isto prejudica o desempenho da rede. Os autores introduziram a ideia de um estrangulamento linear onde a última convolução de um bloco residual tem uma saída linear antes de ser adicionado às ativações iniciais [26].

B.3 ReLU6

Para finalizar a MobileNetV2, há duas outras operações que são necessárias. A primeira é acrescentar a *Batch Normalization* por trás de cada camada convolutiva [26, 64].

A segunda adição é utilizar ReLU6 em vez de ReLU, que limita o valor a um máximo de 6 ativações. A ativação é linear desde que se situe entre 0 e 6 [26].

Isto é útil para quando se trata de inferência com ponto fixo. Deste modo, limita a informação à esquerda do ponto decimal a 3 bits, o que significa que se obtém uma precisão garantida à direita do ponto decimal [26].

Anexo C

Programas em Python

Código C.1: MobilenetV2-UNET em Python [54]

```
1 def build_mobilenetv2_unet(input_shape):    ## (512, 512, 3)
2     """ Input """
3     inputs = Input(shape=input_shape)
4
5     """ Pre-trained MobileNetV2 """
6     encoder = MobileNetV2(include_top=False, weights="imagenet"
7         ,
8         input_tensor=inputs, alpha=0.5)
9     encoder.summary()
10
11     """ Encoder """
12     s1 = encoder.get_layer("input_1").output    ##
13         (512 x 512)
14     s2 = encoder.get_layer("block_1_expand_relu").output    ##
15         (256 x 256)
16     s3 = encoder.get_layer("block_3_expand_relu").output    ##
17         (128 x 128)
18     s4 = encoder.get_layer("block_6_expand_relu").output    ##
19         (64 x 64)
20
21     """ Bridge """
22     b1 = encoder.get_layer("block_13_expand_relu").output    ##
23         (32 x 32)
24
25     """ Decoder """
26     d1 = decoder_block(b1, s4, 512)    ##
27         (64 x 64)
28     d2 = decoder_block(d1, s3, 256)    ##
29         (128 x 128)
```

```

22     d3 = decoder_block(d2, s2, 128)                                ##
        (256 x 256)
23     d4 = decoder_block(d3, s1, 64)                                ##
        (512 x 512)
24
25     """ Output """
26     outputs = Conv2D(1, 1, padding="same", activation="sigmoid"
        )(d4)
27
28     model = Model(inputs, outputs, name="MobileNetV2_U-Net")
29     return model

```

Código C.2: Inception-ResNet-V2-UNET em Python [54]

```

1  def build_inception_resnetv2_unet(input_shape):
2      """ Input """
3      inputs = Input(input_shape)
4
5      """ Pre-trained InceptionResNetV2 Model """
6      encoder = InceptionResNetV2(include_top=False, weights="
        imagenet", input_tensor=inputs)
7
8      """ Encoder """
9      s1 = encoder.get_layer("input_1").output                ## (512
        x 512)
10     s2 = encoder.get_layer("activation").output              ## (255
        x 255)
11     s2 = ZeroPadding2D(( (1, 0), (1, 0) ))(s2)              ## (256
        x 256)
12     s3 = encoder.get_layer("activation_3").output            ## (126
        x 126)
13     s3 = ZeroPadding2D((1, 1))(s3)                          ## (128
        x 128)
14     s4 = encoder.get_layer("activation_74").output           ## (61
        x 61)
15     s4 = ZeroPadding2D(( (2, 1), (2, 1) ))(s4)              ## (64
        x 64)
16
17     """ Bridge """
18     b1 = encoder.get_layer("activation_161").output          ## (30
        x 30)
19     b1 = ZeroPadding2D((1, 1))(b1)                          ## (32
        x 32)
20
21     """ Decoder """
22     d1 = decoder_block(b1, s4, 512)                          ## (64
        x 64)

```

```

23     d2 = decoder_block(d1, s3, 256)                ## (128
        x 128)
24     d3 = decoder_block(d2, s2, 128)                ## (256
        x 256)
25     d4 = decoder_block(d3, s1, 64)                 ## (512
        x 512)
26
27     """ Output """
28     outputs = Conv2D(1, 1, padding="same", activation="sigmoid"
        )(d4)
29
30     model = Model(inputs, outputs, name="InceptionResNetV2_U-
        Net")
31     return model

```

Código C.3: Função *calc_spatials()* utilizada para calcular as coordenadas espaciais [60]

```

1  # roi has to be list of ints
2  def calc_spatials(self, depthFrame, roi, averaging_method=np.
    mean):
3      roi = self._check_input(roi, depthFrame) # If point was
        passed, convert it to ROI
4      xmin, ymin, xmax, ymax = roi
5
6      # Calculate the average depth in the ROI.
7      depthROI = depthFrame[ymin:ymax, xmin:xmax]
8      inRange = (self.THRESH_LOW <= depthROI) & (depthROI <= self
        .THRESH_HIGH)
9
10     averageDepth = averaging_method(depthROI[inRange])
11
12     centroid = { # Get centroid of the ROI
13         'x': int((xmax + xmin) / 2),
14         'y': int((ymax + ymin) / 2)
15     }
16
17     midW = int(depthFrame.shape[1] / 2) # middle of the depth
        img width
18     midH = int(depthFrame.shape[0] / 2) # middle of the depth
        img height
19     bb_x_pos = centroid['x'] - midW
20     bb_y_pos = centroid['y'] - midH
21
22     angle_x = self._calc_angle(depthFrame, bb_x_pos)
23     angle_y = self._calc_angle(depthFrame, bb_y_pos)
24
25     spatials = {
26         'z': averageDepth,

```

```

27         'x': averageDepth * math.tan(angle_x),
28         'y': -averageDepth * math.tan(angle_y)
29     }
30     return spatial, centroid

```

Código C.4: Algoritmo utilizado para monitorizar a força

```

1  import argparse
2  import logging
3  import sys
4  sys.path.append('.')
5  import rtde.rtde as rtde
6  import rtde.rtde_config as rtde_config
7  import time
8
9  # parameters
10 parser = argparse.ArgumentParser()
11 parser.add_argument('--host', default='169.254.249.12', help='
    name of host to connect to (localhost)')
12 parser.add_argument('--port', type=int, default=30004, help='
    port number (30004)')
13 parser.add_argument('--samples', type=int, default=0, help='
    number of samples to record')
14 parser.add_argument('--frequency', type=int, default=125, help='
    the sampling frequency in Herz')
15 parser.add_argument('--config', default='record_configuration.
    xml', help='data configuration file to use (
    record_configuration.xml)')
16 parser.add_argument("--verbose", help="increase output
    verbosity", action="store_true")
17 parser.add_argument("--buffered", help="Use buffered receive
    which doesn't skip data", action="store_true")
18 parser.add_argument("--binary", help="save the data in binary
    format", action="store_true")
19 args = parser.parse_args()
20
21 if args.verbose:
22     logging.basicConfig(level=logging.INFO)
23
24 conf = rtde_config.ConfigFile(args.config)
25 output_names, output_types = conf.get_recipe('out')
26
27
28 con = rtde.RTDE(args.host, args.port)
29 con.connect()
30
31 # settings
32 con.get_controller_version()

```

```

33 con.send_output_setup(output_names, output_types, frequency=
    args.frequency)
34 con.send_start()
35
36
37 i=0
38 keep_running=True
39 for i in range(1):
40     if args.buffered:
41         state = con.receive_buffered(args.binary)
42     else:
43         state = con.receive(args.binary)
44     if state is not None:
45         Fx_old,Fy_old,Fz_old,Tx_old,Ty_old,Tz_old = state.
            actual_TCP_force
46         date_and_time = state.timestamp
47         i += 1
48         print("Force ["+str(Fx_old)+", "+str(Fy_old)+", "+str(
            Fz_old)+"] N, Torque ["+str(Tx_old)+", "+str(Ty_old)
            +", "+str(Tz_old)+"] Nm")
49         print("\n")
50         time.sleep(1)
51
52
53
54 Fx_new,Fy_new,Fz_new,Tx_new,Ty_new,Tz_new=Fx_old,Fy_old,Fz_old,
    Tx_old,Ty_old,Tz_old
55 i=0
56 keep_running=True
57 while(Fx_new<Fx_old+3):
58     if args.buffered:
59         state = con.receive_buffered(args.binary)
60     else:
61         state = con.receive(args.binary)
62     if state is not None:
63         Fx_new,Fy_new,Fz_new,Tx_new,Ty_new,Tz_new = state.
            actual_TCP_force
64         date_and_time = state.timestamp
65         i += 1
66         print(" Force: ["+str(Fx_new)+", "+str(Fy_new)+", "+str
            (Fz_new)+"] N")
67         print("\n")
68         s.send(("move1(pose_add(get_forward_kin(), p[-0.005,
            0.0, 0.0, 0, 0, 0]), a=0.2, v=0.2)" +"\n").encode('
            utf8'))
69         time.sleep(1)
70

```



```
71 | con.send_pause()  
72 | con.disconnect()
```