Recognition of objects to grasp and Neuro-Prosthesis control

Attila Fejer

DUAL DEGREE THESIS PRESENTED

TO OBTAIN THE DEGREE OF

DOCTOR OF PHILOSOPHY OF THE PÁZMÁNY PÉTER CATHOLIC UNIVERSITY

AND

THE UNIVERSITY OF BORDEAUX

ROSKA TAMÁS DOCTORAL SCHOOL OF SCIENCES AND TECHNOLOGY

DOCTORAL SCHOOL OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALTY: COMPUTER SCIENCE

by

Attila FEJÉR

Recognition of objects to grasp and Neuro-Prosthesis control

Date of the defence: 25/11/2022

Before the review panel composed of:

Jenny	Benois-Pineau	Pr.	LaBRI, Université de Bordeaux	Co-director of the thesis
Péter	Szolgay	Pr.	Pázmány Péter Catholic University	Co-director of the thesis
Aymar	de Rugy	Dr.	INCIA, Université de Bordeaux	Examiner
Christophe	Garcia	Pr.	INSA de Lyon	Reviewer
Alexandre	Benoit	Pr.	University of Monblanc-Pays de Savoie	Examiner
László	Czúni	Dr.	Pannon University	Reviewer
Zsolt	Vörösházi	Dr.	Pannon University	Examiner
Janka	Hatvani	Dr.	Pázmány Péter Catholic University	Examiner

The jury is chaired by: Pr. Alexandre BENOIT

– 2022 –

Title

Recognition of objects to grasp and Neuro-Prosthesis control

Abstract

The goal of the present PhD research is to provide assistance to upper-limb amputees wearing robotic prosthetic arms. Classical myoelectric control of neuro-prostheses is limited in case of severe amputation. Computer vision approach and gaze information can provide a valuable indication for prosthetic arm control identifying and localizing an object to grasp.

The prosthetic arm controlling mechanism has to be wearable, fast enough for real-time processing, and mobile. Thus we propose an embedded Hardware-Software FPGA implementation of the gaze-driven object recognition method with a Gaze-Driven CNN developed for natural cluttered video scenes.

To achieve the research goal, we propose different algorithmic solutions adapted to FPGA. The whole embedded system allows for recognition and localization of the object-to-grasp in the 3D space.

To decide which block in the whole algorithmic chain needs to be accelerated on FPGA, and which can be processed by an embedded CPU, we first conduct complexity analysis in terms of computational time for the hybridization choices.

The acquisition device is Tobii eye-tracker glasses which the amputee is supposed to wear in an ergonomic scenario of vision-assisted neuroprosthetic control. Due to the distractors, filtering of gaze points with motion compensation from the past to the present frame is needed. To do this, the Scale Invariant Feature Transform (SIFT) keypoints are extracted in every frame and serve for homography estimation between frames. However, our measurements show that SIFT computational time is too high using just an embedded CPU. So we adopted a SIFT keypoint extractor to an FPGA having proposed a simplified SIFT extractor due to the FPGA resource restrictions. Our solution was compared to the fully SW implemented OpenSIFT. The experiments show that the two implementations yield practically the same result. Our implementation can process a 480px x 480 px image at 135 frames/second on the Xilinx ZCU 102 FPGA board.

To eliminate the outlier gaze points corresponding to distractors which are projected to the current frame with the estimated homography, DBScan clustering is used. The fixated gaze point on the object to grasp is predicted with the Kernel Density Estimation. The estimated gaze point and the current frame are the input of the Gaze Driven CNN. Its goal is to recognize the object and precisely localize it in the video frame. The original CPU implementation of it contains three modules: Resnet50, Faster R-CNN and Multiple Instance Learning. Our Complexity analysis shows that the Resnet50 computational time is too high for the embedded CPU. Gaze-Driven CNN FPGA optimized version is thus built in the following modules: the Resnet50 (FPGA), which extracts the features from a frame, the Reduction

layer (CPU) to simplify the feature tensor, the Faster R-CNN (CPU) to estimate the grasped object location, and the Multiple Instance Learning (CPU) to predict the grasped object type. The FPGA accelerated Resnet50 can process more than 30 fps on ZCU102.

The depth is estimated based on the plane equation of the 3 closest gaze points to the current frame gaze point.

To validate our approach we have conducted our experiments on a real-world Open Source cluttered Grasping-in-the-Wild (GITW) dataset recorded in kitchens. The subjects are grasping common objects such as a pan, a bowl, etc. The dataset contains 16 different objects in 7 kitchens and 404 egocentric videos.

Our experiments show that the proposed hybrid solution allows achieving 8.5 fps frame rate without losing the accuracy of the original CPU implemented approach. Further acceleration by pipelining is achievable and remains in the perspective of the present work.

Keywords

deep learning, FPGA, Object Recognition

Titre

Reconnaissance des objets à saisir et contrôle d'un bras robotique pour l'assistance aux porteurs des neuro-prothèses

Résumé

L'objectif de la recherche doctorale est de fournir une assistance aux amputés des membres supérieurs portant des prothèses robotisés. Le contrôle myoélectrique classique des neuro-prothèses est limité en cas d'amputation sévère. L'approche de la vision par ordinateur et les informations sur le regard peuvent fournir une indication pour le contrôle de la prothèse en identifiant et en localisant un objet à saisir.

L'outil de contrôle doit être portable, suffisamment rapide pour un traitement en temps réel, et mobile. Nous proposons donc une implémentation matérielle et logicielle FPGA de la méthode de reconnaissance d'objets, avec un CNN guidé par le regard, développée pour les scènes vidéo naturelles encombrées.

Pour atteindre l'objectif de recherche, nous proposons différentes solutions algorithmiques adaptées au FPGA. L'ensemble du système embarqué permet la reconnaissance et la localisation de l'objet à saisir dans l'espace 3D.

Pour décider quel bloc de la chaîne algorithmique doit être accéléré sur FPGA, et lequel reste sur le CPU embarqué, nous effectuons d'abord une analyse de complexité en temps de calcul.

Le dispositif d'acquisition est constitué des lunettes eye-tracker Tobii que l'amputé est censé porter dans un scénario ergonomique de contrôle neuroprothétique assisté par la vision. En raison des distracteurs, il est nécessaire de filtrer les points de regard avec une compensation de mouvement des images antérieures à l'image courante. Pour ce faire, les points clés SIFT (Scale Invariant Feature Transform) sont extraits dans chaque image et servent à l'estimation de l'homographie entre les images. Cependant, nos mesures montrent que le temps de calcul de SIFT est trop élevé sur le CPU embarqué. Nous avons donc adapté un extracteur de points clés SIFT sur un FPGA en proposant un algorithme simplifié en raison des restrictions de ressources du FPGA. Notre solution a été comparée à l'implémentation entièrement logicielle d'OpenSIFT. Les expériences montrent que les deux implémentations donnent pratiquement le même résultat. Notre implémentation peut traiter une image de 480 px x 480 px à 135 images/seconde sur la carte FPGA Xilinx ZCU 102.

Pour éliminer les points de regard aberrants correspondant aux distracteurs, le clustering des points (DBScan) est appliqué. Le point de fixation du regard sur l'objet à saisir est prédit à l'aide de l'estimateur à noyau. Le point de regard estimé et l'image courante constituent l'entrée du CNN piloté par le regard. Son objectif est de reconnaître l'objet et de le localiser précisément dans l'image vidéo. L'implémentation originale du CPU contient trois modules : Resnet50, Faster R-CNN et Multiple Instance Learning. Notre analyse de la complexité montre que le temps de calcul de Resnet50 est trop élevé pour le CPU embarqué. La version FPGA optimisée du CNN est donc construite dans les modules suivants : le Resnet50

(FPGA), qui extrait les caractéristiques d'une image, la couche de réduction (CPU) pour simplifier le tenseur de caractéristiques, le Faster R-CNN (CPU) pour estimer la localisation de l'objet, et le Multiple Instance Learning (CPU) pour prédire le type d'objet saisi. Le Resnet50 accéléré par FPGA peut traiter plus de 30 images par seconde sur ZCU102.

La profondeur est estimée sur la base de l'équation du plan des 3 points de regard les plus proches du point de regard de l'image actuelle.

Pour valider notre approche, nous avons mené nos expériences sur un ensemble de données Open Source, Grasping-in-the-Wild (GITW), enregistrées dans des cuisines. Le jeu de données contient 16 objets différents dans 7 cuisines et 404 vidéos égocentriques.

Nos expériences montrent que la solution hybride proposée permet d'atteindre une fréquence d'images de 8,5 fps sans perdre la précision de l'approche originale implémentée sur CPU. Une accélération supplémentaire par pipelining est réalisable et reste dans la perspective du présent travail.

Mots-clés

l'apprentissage en profondeur, FPGA, reconnaissance d'objet

Table of contents

List of Figures **1**

List of Tables **3**

Acknowledgements **7**

List of acronyms **9**

Summary of the PhD dissertation in English **13**

Résumé de la thèse de doctorat en français **18**

1 Introduction **24**

2 State-of-the-Art in visual scene analysis for neuroprostheses controls and FPGA implementation **28**

2.1 Introduction 28

2.2 Neuroprosthesis control based on visual and gaze information 30

2.3 Object recognition in visual scenes 34

2.4 FPGA implementation of visual scenes analysis methods 37

2.5 Comparing different hardware for an embedded system from computer vision point of view 44

2.6 Existing hybrid HW/SW solutions on FPGA 45

2.7 Conclusion 47

3 Analysis of object-to-grasp recognition in view of FPGA based implementation 50

3.1 Introduction . . . 50

3.2 Visual servoing of upper-limb prosthesis scenario . . . 52

3.3 Object-to-grasp recognition approach . . . 53

3.4 Target FPGA board . . . 64

3.5 Critical analysis of computation complexity . . . 74

3.6 Conclusion . . . 81

4 Hybrid solutions for SIFT detector implementation 83

4.1 Introduction . . . 83

4.2 Base-Line SIFT algorithm . . . 84

4.3 Hybrid SIFT implementation . . . 90

4.4 Experiments and results . . . 97

4.5 Conclusion . . . 110

5 Optimized implementations of preprocessing steps 111

5.1 Introduction . . . 111

5.2 Gaze point alignment . . . 112

5.3 Gaze point noise reduction . . . 118

5.4 Depth estimation . . . 122

5.5 Results . . . 125

5.6 Conclusion . . . 132

6 Hybrid solutions for object recognition with Gaze-Driven CNN 134

6.1 Introduction . . . 134

6.2 Gaze-Driven CNN implementation . . . 135

6.3 Results . . . 140

6.4 Conclusion 147

7 Conclusion **149**

7.1 New scientific results 149

7.2 Perspectives 153

Bibliography **155**

A Publications related to the thesis **173**

A.1 Journal publications 173

A.2 Conference publications 173

List of Figures

2.1 The bimodal information architecture (Fukuda et al., 2021). 32

2.2 Example of the residual block in the ResNet. 34

3.1 Example of the Tobii 2 glass camera usage. 52

3.2 The prosthetic arm visually guided system. 54

3.3 Gaze-driven CNN for object recognition. Upper part: Preprocessing pipeline. Lower part: the Gaze Driven CNN on object proposals (González-Díaz et al., 2019). 56

3.4 Examples of objects of the GITW(LaBRI, 2016) dataset. 60

3.4 Example object of the GITW LaBRI (2016) dataset. 61

3.5 Example of the Bowl subset. It contains different bowls in different kitchens. 63

3.6 Example of non-cluttered and cluttered environment. 64

3.7 Basic FPGA architecture. 64

3.8 Vitis AI quantization process Xilinx (2022c). 74

4.1 The main steps of the SIFT(Lowe, 2004) algorithm. 85

4.2 The block diagram of our proposed architecture. The SIFT detector part contains the GFDG computation, SSE, NMS and Edge Detection modules. The SIFT descriptor part contains the orientation assignment and other descriptor computation steps. 91

4.3 Gaussian Filtering and Difference of Gaussians computation (GFDG). The red rectangle is the current computed pixel. The green rectangle is the current input pixel from the input stream. Neighbours of the currently computed pixel are stored in the input image window. Two lines of the most recently used pixels are stored in the "temporary row delay arrays". Gaussian blurred image and the Difference of Gaussian image are the outputs of the module. 93

4.4 Calculate the local extremas in one scale (Lowe, 2004). 94

4.5 Comparison between the OpenSIFT (Hess, 2010) and the FPGA implementation detected keypoints. Y-axis is the number of the extracted keypoints, X-axis is the frame number. FPGA-detected keypoints are in blue, CPU-detected keypoints are in red. 99

4.6 Visual comparison of detected sets of keypoints. On the left: OpenSIFT (Hess, 2010) extracted keypoints. On the right: the FPGA result. 99

5.1 Example of bowl place 4 subject 2 gaze point alignment. The points are the gaze points. 113

5.2 Chain of the DBSCAN algorithm. The two parameters are the ϵ and the minPts. 120

5.3 Bowl Place 5 Subject 2 DBSCAN parameters: $\epsilon = 0.01$, $min\_samples = 3$, good gaze points colour blue, outlier gaze points: green. 121

5.4 Example of bowl place 4 subject 2 KDE gaze point estimation. The points are the gaze points and the white point is the estimated gaze point. 122

5.5 Depth estimation chain. 123

6.1 Gaze-driven, object-recognition CNN, where CH is the number of output channels of the Reduction Layer. 136

6.2 Example of “Bowl place 1, subject 1", (GITW) generated bounding box. The bounding boxes are generated around the red bowl. 137

6.3 ResNet50 res4f architecture, shown with the residual units, the size of the filters and the outputs of each convolutional layer. Downsampling is performed by conv2_1, conv3_1, and conv4_1 with a stride of 2. . 137

6.4 Training accuracy (in red) and loss (in blue) during 30 epochs with different number of channels due to Reduction Layer. 144

List of Tables

3.1 Bowl subdataset overview. 62

3.2 Xilinx Zynq UltraScale+ ZCU102 programmable logic resources. . . . 66

3.3 The average computational time measurement of the whole system on a regular computer and an embedded system. The goal is a system which can process a frame less than 100ms and each module maximum computation time should be less than 40ms. The SIFT, KDE estimation, ResNet50, and the Faster R-CNN computation times are too high for real-time processing. 76

3.4 Hybridization of preliminary steps in the pipeline, which contains two main blocks: Gaze Point Alignment Block and Gaze Point Noise Reduction Block and its submodules. 80

3.5 Hybridization of the gaze-driven CNN. 80

4.1 The default parameter list in experiments. 101

4.2 Comparison of the number of keypoints in case of OpenSIFT (CPU)(Hess, 2010) and our FPGA implementations. P is the Precision and R is the Recall. 102

4.3 The average recall in case of different NMS threshold values. The red column is when the NMS threshold was set to 0.01. The average recall is higher than 0.9 when the threshold is 0.01. 104

4.4 The average precision in case of the different NMS threshold values. The red column is when the NMS threshold was set to 0.01. The average precision is higher than 0.81 when the threshold 0.01. 104

4.5 The average number of KPs extracted in case of different NMS threshold values. The red column shows when the NMS threshold is set to 0.01. The number of extracted KPs are computed just using the first octave of the Gaussian pyramids. 105

4.6 1 octave resource usages on Xilinx ZCU102 FPGA Board from Vivado 2018.3 (Xilinx, 2018b) when the default parameters in Table 4.1 has been used. 106

4.7 Resource usage estimation of the main modules of the system based on the Vivado HLS report. 106

4.8 Comparison of SIFT keypoint extraction in different platforms. The Intel Xeon E5-2620 and ARM Cortex-A53 are used the OpenSIFT(Hess, 2010) to extract the keypoint. The Xilinx UltraScale+ ZCU102 are used our FPGA implementation to extract the keypoint. All three platform which used OpenSIFT are used with the default parameters (Table 4.1). On the CMOS Vision Sensor, only the time for Gaussian Pyramid calculation is reported. In the case of NVIDIA Jetson TX 2 the computation time is based on the report of da Costa Barreiros (2020). 108

4.9 Comparison of our work with different FPGA implementations. . . . 110

5.1 Comparison between the Intel i5 7300HQ and the Xilinx ZCU102 ARM CORTEX A53 in the whole gaze alignment chain. 127

5.2 Comparison in processing time of kernel density estimation module between the Intel i5 7300HQ and the Xilinx ZCU102 ARM CORTEX A53. 128

5.3 Average computational time of the KDE after 100 iterations on the Xilinx ZCU 102 Embedded ARM CPU. The gaze points number is 10 and the size of the area is given. 129

5.4 DBSCAN without setting the gaze point maximum number to 10. . . 130

5.5 DBSCAN with gaze point maximum number set to 10. 130

5.6 KDE + DBSCAN with removed outliers (Gaze Points area size <4900 px), DBSCAN parameters: ϵ=0.05, $min\_samples$=5. 131

6.1 Measurements of the gaze-driven, object-recognition CNN in the Intel i5 7300 CPU. The first column contains the reaming number of channels after the Reduction Layer. Each column shows the elapsed time during the computation in milliseconds. 142

6.2 Measurements of the gaze-driven, object-recognition CNN in the ARM A53 CPU. The first column contains the remaining number of the channels after the Reduction Layer. Each column shows the elapsed time during the computation in milliseconds. 143

6.3 The results of the training and testing after 30 epochs. 145

6.4 Comparison of different object recognition CNNs. All the measurements were taken by Vitis AI 1.4. The gaze-driven, object-recognition CNN used 128 channels in the Reduction Layer. 145

6.5 The average computational time measurement of the whole system on different hardware. The ResNet50 number of channels is 128. . . . 146

Acknowledgements

This thesis was facilitated by the co-tutelle agreement between the Pázmány Péter Catholic University of Budapest and the University of Bordeaux – in particular with Laboratoire Bordelais de Recherche en Informatique (LaBRI).

I think doing a PhD alone is impossible, and I am lucky because I have met the right people who helped me during this period.

I would like to thank my supervisors Jenny Benois-Pineau, Péter Szolgay, Aymar de Rugy and Zoltán Nagy.

I have learnt a lot from Zoltán Nagy about the developing an algorithm on FPGA. He also helped me with teaching, and he gave me insightful tips on writing scientific papers. He was available at any time and answered all my questions I had during my Phd journey.

I am extremely thankful that I had the opportunity to do my PhD studies at the University of Bordeaux with Jenny Benois-Pineau. She made my last few years easier by not only guiding through my studies but also helping me with my integration in Bordeaux. She helped me writing the conference and journal papers, while also was available in times where I had difficulties and i required some help.

Aymar de Rugy helped me with his expert knowledge about the neuroprosthetic arm and reviewed my English in the scientific papers.

Péter Szolgay gave me many insightful tips on how to organize my time when I am writing a scientific paper. He also helped me to earn scholarships during my Phd.

I acknowledge a grant from the Institut Français en Hongrie.

I am grateful for the help of both doctoral schools, the Roska Tamás Doctoral School and the École doctorale Mathématiques et informatique de l'université de Bordeaux. I would like to thank to Tivadarné Vida (Katinka néni) the help of the administrative tasks.

I appreciate the scientific remarks of my thesis follow-up committee Pascal Desbarats and Akka Zemmari.

Big thanks to my friends and collages in LaBRI, Association de la Formation Doctorale d'Informatique de Bordeaux, and Pázmány Péter Catholic University, with whom I work, drink coffee and tea, talk and discuss things in the kitchen, travel, and helping me during the COVID-19 situation, present with, and made my PhD experience rewarding: Pierre-Etienne Martin, Dániel Hajtó, Miklós Tóth, András Attila Sulyok, Zoltán Horváth, János Vincze, Thinhinane Yebda, Thi Thi Trang Ngo, Karim Adherhal, Karim Alami, Soumyajit Paul, Varun Ramanathan, Sougata Bosa, Tidiane Sylla, Abdennour Rachedi, Badreddine Yacine Yacheur, Miltiadis Poursanidis.

I would like to say a big thank you to Dániel Hajtó for being my friend and for his patience to explain me things, travelling with me, and share his opinion with me.

Also, thanks to My Family for encouraging me and believing me throughout my university study period even in the darkest times. Special thanks to My Brother Andris who helped me endlessly correct my grammatical mistakes and my baroque-style sentences.

List of acronyms

AR augmented reality 29, 32, 33

AXI Advanced eXtensible Interface 68–70

BB bounding box 3, 35, 54, 55, 57, 58, 62, 75, 76, 122, 131–133, 135–137, 139, 140, 146, 147, 150, 151

BRAM block RAM 66, 69, 106, 107, 152

BRIEF Binary Robust Independent Elementary Feature 33, 40

CNN convolutional neural network 1, 3, 4, 6, 17, 23, 26, 27, 31, 32, 34, 35, 37, 44, 45, 48, 49, 51, 55, 56, 58, 78–80, 110, 111, 133–136, 142, 143, 145, 147, 149, 150

DBSCAN Density-Based Spatial Clustering of Applications with Noise 2, 6, 78, 112, 118–121, 125, 129–133, 146

DNN deep neural network 24, 26

DoG Difference of Gaussians 41, 86, 87, 91, 92, 94–97

DSP digital signal processing 65, 66, 106, 107, 152

ED Edge Detection 1, 90, 91, 106, 107

EMG electromyographic 24, 31–33

FF flip-flop 65, 66, 106, 107, 152

FINN Framework for Fast, Scalable Binarized Neural Network Inference 30, 37, 38

FLANN Fast Library for Approximate Nearest Neighbours 27, 33, 76, 112, 114, 116, 125, 126, 132, 146

FP Feautre Pyramid 139

FPGA Field-Programmable Gate Array 1, 2, 4, 5, 7, 17, 22, 23, 25–27, 30, 37–40, 42–49, 51, 64–67, 71, 72, 77–81, 83, 84, 89, 90, 92, 95, 96, 98–103, 106–110, 132, 133, 135, 142, 146–153

FPN Feature Pyramid Network 139

fps frames per second 41–43, 53, 55, 61, 75, 110, 126, 128, 129, 132, 133, 141, 142, 145–148, 150, 151

GFDG Gaussian Filtering and Difference of Gaussians computation 1, 2, 90–94, 98, 106, 107

GITW Grasping In The Wild 1, 36, 48, 51, 58, 60–62, 81, 90, 97, 115, 130–132, 141, 145

GP gaze point 6, 78, 130–132

HW/SW hardware/software 46, 47, 77, 89, 90, 110, 152

IP intellectual property 68–70, 72, 73, 106

KDE Kernel Density Estimation 2, 4, 6, 27, 56, 75–78, 80, 112, 118, 119, 121, 122, 125, 128, 129, 131–133, 145, 146

KP keypoint 2, 4, 5, 26, 27, 40, 41, 43, 48, 75, 77, 78, 83, 85–88, 90, 94–97, 99–103, 105, 108, 110–113, 116, 125–127, 132, 151–153

LaBRI Laboratoire Bordelais de Recherche en Informatique 7, 8, 26, 150

LUT lookup table 65, 66, 69, 106, 107, 152

mAP mean average precision 36, 48

MIL Multiple Instance Learning 27, 58, 75, 76, 80, 135, 137, 140, 146, 147, 150

MPSoC multiprocessor system on a chip 45, 65–67, 73, 81

ms millisecond 4, 6, 25, 32, 41, 43, 55, 75–77, 90, 114, 127–129, 131, 132, 142, 143, 146, 148, 151, 152, 154

MSE mean squared error 50

mW milliwatt 107, 108

NMS Non-maximum Supression 1, 4, 5, 89–91, 95, 101–107, 152

ORB Oriented FAST and Rotated BRIEF 33, 40, 48, 83

PL programmable logic 4, 66, 67, 69, 71, 81

PS processing system 66, 67, 69, 80, 81

R-CNN Region-based Convolutional Neural Network 4, 27, 30, 34–36, 48, 55, 57, 72, 75–77, 80, 81, 134–137, 139, 140, 146, 147, 150, 151

RANSAC RANdom SAmple Consensus 33, 42, 112, 117, 132

ResNet Residual Network 1, 3, 4, 6, 27, 30, 34–36, 48, 57, 72, 73, 75–77, 80, 135–138, 142, 146, 147, 150, 151

RL Reduction Layer 3, 6, 27, 80, 135, 136, 138, 139, 142–147, 150, 151

ROI region of interest 57, 136, 139, 142, 143

RTL register-transfer level 67–70, 81

SIFT Scale-Invariant Feature Transform 1, 4, 5, 17, 23, 26, 27, 37, 40–43, 48, 63, 75–78, 80, 81, 83–85, 87–91, 95–97, 99, 100, 103, 106–114, 125–127, 132, 133, 146, 149–153

SoC system on chip 44, 45, 70–72

SOTA state-of-the-art 25, 26, 28, 30, 34, 109, 145

SSD Single-Stage Object Detection 30, 34, 36, 48, 72, 73, 134, 145

SSE Scale-Space Extrema Search 1, 90–92, 106, 107

SURF Speeded-Up Robust Features 40, 48, 83

VGG Visual Geometry Group 30, 34–36, 48, 72, 73

VR virtual reality 29–31, 33

VTA Versatile Tensor Accelerator 30, 37, 39

W Watt 41, 67, 75, 77, 107, 108, 110, 146, 153

YOLO You Only Look Once 30, 34, 36, 48, 72, 134, 145

Summary of the PhD dissertation in English

Today, the use of a prosthetic arm has become an accessible and convenient tool for people who have lost their upper limbs due to an unfortunate accident. Technological advancements lead to the development of more comfortable and suitable prosthetic, making the user life easier and more convenient. Our vision is to further enhance the quality of arm prosthetic using visual information which aids its performance tackling everyday problems. In order to keep the lightweight of this system, our visual controlling mechanism must be suitable to wear and allow real-time processing while it should not limit the mobility of the artificial limb. Carefully examining these requirements, we have decided to develop the controlling mechanism in FPGA. This is an open problem and there are intensive research activities in this area.

The visual control program was proposed in previous works of LABRI and has several individual parts. It works on a regular computer. Our goal is to accelerate those algorithms to achieve the real-time processing speed.

It is a complex computer vision, image processing and a robotic problem, because we have to find the object, which the user wants to grasp. This part is the computer vision and image processing task. We also want to control the robotic arm to help the user to grasp this object, which is the robotic problem.

In this dissertation, we discuss the State-of-the-art methods (SOTA) of the visual-guided neuroprostheses control, the object recognition and the FPGA im-

plementation of visual scenes analysis methods. Thus, our system is compared to other SOTA solutions.

Hence, we propose a visual guided system to help control a neuroprosthesis arm.

The Tobii eye-tracker camera is the acquisition device. The user can wear this camera as a glass. The device can record an egocentric view video. The recorded video resolution is Full HD (1920 $\times$ 1080) with 25 fps. It can also record the user gaze fixations showing the location of the user' gaze looking in a frame, and the distance between the object of interest and the camera in millimeter precision.

"Grasping-in-the-wild" dataset developed in LABRI was used in our research. This dataset contains real-world video recordings in different kitchens, so the recordings vary. It is an Open dataset freely available for research at Nakala CNRS server. The videos are recorded in an egocentric view when a user is grasping a kitchen equipment. There are 16 different kinds kitchen equipment/objects in this dataset such as bowl, can of coke, milk bottle, and so on. The dataset contains 404 videos with the average duration around 10 seconds. The videos have different complexity from the computer vision point of view, but all remain "natural" and thus complex for an automatic analysis. The environment can be cluttered, because there are multiple objects in a real-world kitchen. Thus, the equipment can be transparent, such as a glass or bottle. The dataset contains the egocentric view videos and the user gaze fixation coordinates.

Our experiments show that a pure software implementation of algorithms is not fast enough to control a neuroprosthetic arm in real time. An ideal solution for this problem should be able to process the data in real-time. The power consumption of the device has to be very low, as the user has to wear it during his natural activities. The Field-Programmable Gate Arrays (FPGA) satisfy those requirements.

The Xilinx Zynq UltraScale+ MPSoC ZCU102 board has been chosen as a target

development device. This board is ideal to accelerate computer vision algorithms.

The XCZU9EG FPGA device on the ZCU102 board has a Processing System (PS) and a Programmable Logic (PL) part. The PS part has a quad-core ARM Cortex-A53, dual-core Cortex-R5F real-time processors, and a Mali-400 MP2 graphics processing unit. The Cortex-A53 is an Application Processing Unit (APU) to run OS and general purpose applications. The ZCU102 has a Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC chip. The Cortex-A53 is an ARM v8 architecture-based 64-bit quad-core multiprocessing CPU. The Cortex-R5 is a Real-time Processing Unit (RPU) and is based on an ARM v7 architecture 32-bit RPU with a dedicated tightly coupled memory (TCM). The Mali-400 is a graphics processing unit with pixel and geometry processor and 64 KB L2 cache.

PL resources of ZCU102 are the following: it has 912 Block RAM (BRAM), 548160 flip-flops (FF), 2520 digital signal processing (DSP) units, and 274080 Look-up tables (LUT).

To control a prosthetic arm for the amputees, we proposed a hybrid hardware-software solution. The system main components are the following: Recording the frames and gaze points with Tobii eye-tracker, gaze points alignment, gaze points noise reduction, bounding boxes generation around the noise reduced gaze points, object detection with the Gaze Driven CNN. More information about each component could be found in the next paragraphs.

The Tobii eye-tracker records the user gaze points and an egocentric video.

User gaze points can contain noise due to the distractors and microsaccades. Thus we wish to collect amm the gaze points in the current frame and estimate the smooth position of the gaze fixation in the current video frame. Thus the next step is the gaze points alignment. We wish to perform it with a homography model; The latter can be built by matching characteristic points of the visual scene. Thus, first

we need to extract the Scale-Invariant Feature Transform (SIFT) keypoints of every consecutive frames in a temporal window we wish to use for estimation of smoothed gaze point position in the current frame. Our measurements show that SIFT Lowe (2004) was one of the slowest parts of our method for the experimental system.

Having the SIFT keypoints the next step is the FLANN matching. There is a reference image which we compare the reference image-10, reference image- ...., reference image-1 images and put them in the same plane. After that we can compute the homography matrix with matched key-points, and project all the gaze points from the temporal window in the current video frame.

In the gaze-point noise reduction, the goal is to filter out the outlier gaze points. The outlier gaze points may be caused by the saccades, and the head movements, when the user tries to find to object or get distracted during the process of grasping. The DBScan clustering removes the outlier gaze points and the Kernel Density Estimation (KDE) uses the remained aligned gaze points and estimate the smoothed location of the gaze point.

Bounding boxes are then generated around the estimated gaze point. The classification of these bounding boxes accordingly to our object taxonomy will give a precised location of the object-to-grasp and its type. There are 9 bounding boxes generated with different scales and sizes where the center point is the estimated gaze point.

The Gaze Driven CNN is predicting the object type and the object location. Inputs of this CNN are the nine bounding boxes and the current video frame.

The object type is necessary. Indeed, in grasping process, it is important to know the shape and the surface quality of the object. In our current work we have not studied the palm opening control as a function of the object shape, but prepared a good basis for this next step of the prosthesis control.

To summarize, in this PhD research, we have developed a full solution for object recognition with Deep NNs in ego-centered video on a hybrid architecture using FPGA. The solution is guided by gaze fixation recordings from the eye-tracker and is designed for visual servoing of the upper limb neuroprostethic arms. In the following, we summarize the contributions.

My scientific results are two-fold:

- 1. I developed a hybrid solution: FPGA-CPU - for object recognition in ego-centered video by a Gaze-Driven CNN.
- 2. As a re-usable part of it I have implemented, on FPGA, a new SIFT detector for pre-processing of gaze data, namely their alignment in the current video frame.

Résumé de la thèse de doctorat en français

Aujourd'hui, l'utilisation d'un bras prothétique est devenue un outil accessible et pratique pour les personnes qui ont perdu leurs membres supérieurs à la suite d'un accident malheureux. Les progrès technologiques conduisent au développement de prothèses plus confortables et plus adaptées, rendant la vie de l'utilisateur plus facile et plus pratique. Notre vision est d'améliorer encore la qualité des prothèses de bras en utilisant des informations visuelles qui aident leurs performances à résoudre les problèmes quotidiens. Afin de conserver la légèreté de ce système, notre mécanisme de contrôle visuel doit être adapté au port et permettre un traitement en temps réel, sans pour autant limiter la mobilité du membre artificiel. En examinant attentivement ces exigences, nous avons décidé de développer le mécanisme de contrôle en FPGA. Il s'agit d'un problème ouvert et unhe recherche intense est menée dans ce domaine.

Une approche de contrôle visuel a été proposé dans des travaux précédents du LABRI. Elle comporte plusieurs étapes. L'implantationfonctionne sur le CPU/GPU d'un ordinateur banalisé. Notre objectif est d'accélérer ces algorithmes pour atteindre la vitesse de traitement en temps réel.

C'est un problème complexe de vision par ordinateur, de traitement d'image et de robotique, car nous devons reconnaître et localiser dans la vidéo, l'objet que l'utilisateur veut saisir. Cette partie est la tâche de vision par ordinateur et de

traitement d'image. Nous voulons également contrôler le bras robotique pour aider l'utilisateur à saisir cet objet, ce qui constitue le problème robotique.

Dans cette thèse, nous discutons des méthodes de l'état de l'art (SOTA) du contrôle des neuroprothèses guidées visuellement, de la reconnaissance des objets et de l'implémentation FPGA des méthodes d'analyse des scènes visuelles. Ainsi, notre système est comparé à d'autres solutions SOTA.

Nous proposons donc un système à guidage visuel pour aider à contrôler un bras robothique de neuroprothèse.

La caméra de l'eye-tracker Tobii est le dispositif d'acquisition. L'utilisateur peut porter cette caméra sur les lunettes. L'appareil peut enregistrer une vidéo en vue égocentrique. La résolution vidéo enregistrée est Full HD (1920 × 1080) avec 25 fps. Il peut également enregistrer les fixations du regard de l'utilisateur en montrant l'emplacement du regard de l'utilisateur dans l'image vidéo, et la distance entre l'objet d'intérêt et la caméra avec la précision millimétrique.

Le jeu de données "Grasping-in-the-wild" développé au LABRI a été utilisé dans notre recherche. Ce jeu de données contient des enregistrements vidéo des scènes réelles dans différentes cuisines, les enregistrements varient. Il s'agit d'un jeu de données ouvert disponible librement pour la recherche sur le serveur Nakala du CNRS. Les vidéos sont enregistrées dans une vue égocentrique lorsqu'un utilisateur saisit un ustensile de cuisine. Il y a 16 différents types d'ustensiles/objets de cuisine dans ce jeu de données, comme un bol, une canette de coca, une bouteille de lait, etc. L'ensemble de données contient 404 vidéos d'une durée moyenne d'environ 10 secondes. Les vidéos ont une complexité différente du point de vue de la vision par ordinateur, mais toutes restent "naturelles" et donc complexes pour une analyse automatique. L'environnement peut être encombré, car il y a de multiples objets dans une cuisine dnas la vie quotidienne. Ainsi, l'obet peut être transparent, comme un

verre ou une bouteille. Le jeu de données contient les vidéos de la vue égocentrique et les coordonnées de fixation du regard de l'utilisateur.

Nos expériences montrent qu'une implémentation purement logicielle des algorithmes n'est pas assez rapide pour contrôler un bras neuroprothétique en temps réel. Une solution idéale pour ce problème devrait être capable de traiter les données en temps réel. La consommation d'énergie du dispositif doit être très faible, car l'utilisateur doit le porter pendant ses activités naturelles. Les réseaux de portes programmables en champ (FPGA) répondent à ces exigences. La carte Xilinx Zynq UltraScale+ MPSoC ZCU102 a été choisie comme dispositif de développement cible. Cette carte est idéale pour accélérer les algorithmes de vision par ordinateur.

Le FPGA XCZU9EG de la carte ZCU102 comporte un système de traitement (PS) et une partie logique programmable (PL). La partie PS comprend un processeur quadricœur Arm Cortex-A53, des processeurs en temps réel Cortex-R5F à double cœur et une unité de traitement graphique Mali-400 MP2. Le Cortex-A53 est une unité de traitement d'application (APU) pour exécuter le système d'exploitation et les applications générales. Le ZCU102 est équipé d'une puce MPSoC Zynq UltraScale+ XCZU9EG-2FFVB1156. Le Cortex-A53 est un processeur multiprocesseur quadricœur 64 bits basé sur l'architecture ARM v8. Le Cortex-R5 est une unité de traitement en temps réel (RPU) et est basé sur une architecture ARM v7 RPU 32-bit avec une mémoire dédiée à couplage serré (TCM). Le Mali-400 est une unité de traitement graphique avec un processeur de pixels et de géométrie et un cache L2 de 64 Ko.

Les ressources PL du ZCU102 sont les suivantes : il dispose de 912 Block RAM (BRAM), 548160 flip-flops (FF), 2520 unités de traitement numérique du signal (DSP) et 274080 Look-up tables (LUT).

Pour contrôler un bras prothétique pour les amputés, nous avons proposé une

solution hybride matériel-logiciel. Les principaux composants du système sont les suivants : Enregistrement des images et des points de regard avec l'eye-tracker Tobii, alignement des points de regard, réduction du bruit des points de regard, génération de boîtes de délimitation autour des points de regard réduits en bruit, détection d'objets avec le Gaze Driven CNN. De plus amples informations sur chaque composant sont disponibles dans les paragraphes suivants.

L'eye-tracker Tobii enregistre les points de regard de l'utilisateur et une vidéo égocentrique.

Les points de regard de l'utilisateur peuvent contenir du bruit dû aux distracteurs et aux microsaccades. Nous souhaitons donc collecter tous les points de regard dans l'image actuelle et estimer la position lissée de la fixation du regard dans l'image vidéo actuelle. L'étape suivante est donc l'alignement des points de regard. Nous souhaitons l'effectuer à l'aide d'un modèle d'homographie ; ce dernier peut être estimé en faisant correspondre des points caractéristiques de la scène visuelle. Ainsi, nous devons d'abord extraire les points caractéristiques SIFT (Scale-Invariant Feature Transform) de chaque image consécutive dans une fenêtre temporelle que nous souhaitons utiliser pour estimer la position du point de regard dans l'image actuelle. Nos mesures montrent que la transformation SIFT est l'une des parties les plus lentes de notre méthode pour le système expérimental. Après avoir obtenu les points clés SIFT, l'étape suivante est la correspondance FLANN. Il y a une image de référence que nous comparons avec l'image de référence-10, l'image de référence-...., l'image de référence-1 et nous les porjetons dans le même plan. Ensuite, nous pouvons calculer la matrice d'homographie avec les points caractéristiques appariés, et projeter toutes les fixations de regard de la fenêtre temporelle dans l'image vidéo actuelle.

Dans la réduction du bruit des points de regard, l'objectif est de filtrer les points

de regard aberrants. Les points de regard aberrants peuvent être causés par les saccades et les mouvements de la tête, lorsque l'utilisateur essaie de trouver un objet ou est distrait pendant le processus de saisie. Le l'algorithme de clustering DBScan élimine les points de regard aberrants. Ensuite l'estimation de la densité par une méthode à noayaux(KDE) utilise les points de regard alignés et estime l'emplacement lissé du point de regard dasn l'image courante.

Des boîtes englobantes sont ensuite générées autour du point de regard estimé. La classification de ces boîtes englobantes selon notre taxonomie d'objets donnera une localisation précise de l'objet à saisir et de son type. Neuf boîtes englobantes sont générées à différentes échelles et tailles, le point central étant le point de regard estimé.

Le CNN piloté par le regard prédit le type et la localisation de l'objet. Les entrées de ce CNN sont les neuf boîtes englobantes et l'image vidéo actuelle.

Le type d'objet est nécessaire. En effet, dans le processus de préhension, il est important de connaître la forme et la qualité de la surface de l'objet. Dans notre travail actuel, nous n'avons pas étudié le contrôle de l'ouverture de la paume en fonction de la forme de l'objet, mais nous avons préparé une bonne base pour cette prochaine étape du contrôle de la prothèse.

Pour résumer, dans cette recherche doctorale, nous avons implementé, sur une architecture hybride qui comprend les FPGA, une solution complète pour la reconnaissance d'objets avec des réseaux profonds dans une vidéo égocentrée. La solution est guidée par les enregistrements de fixation du regard de l'eye-tracker et est conçue pour l'asservissement visuel des bras neuroprosthiques des membres supérieurs. Dans ce qui suit, nous résumons les contributions de notre travail.

Mes résultats scientifiques sont de deux ordres :

- 1. J'ai développé une solution hybride : FPGA-CPU - pour la reconnaissance

d'objets dans une vidéo égocentrique par un CNN guidé par le regard.

- 2. Comme partie réutilisable, j'ai implémenté, sur FPGA, un nouveau détecteur SIFT pour le prétraitement des données du regard, à savoir leur alignement dans l'image vidéo courante.

Chapter 1

Introduction

One of the problems assistive robotics addresses is the production of upper limb prostheses for amputees. Despite great progress in upper limb bionic prostheses, allowing for object-of-interest reaching and grasping, the key remaining issues relate to their control by the operator. To overcome the limitations of traditional control solely based on the electromyographic (EMG) activity of the remaining muscles, promising alternatives consider hybrid systems combining noninvasive motion capture and vision control mentioned in Kanishka Madusanka et al. (2017); Mick et al. (2021). They include camera vision modules that allow for recognition of the subject's intention to grasp an object and assist visual control of prosthetic arms for object reaching and grasping (Han et al., 2020).

The computer vision algorithms which are implemented in these systems comprise the latest object recognition approaches, such as deep neural network (DNN) classifiers and regressors (González-Díaz et al., 2019).

Despite the fact that the visual servoing of robotic arms has been a highly researched area (Hussein, 2015), the application to arm neuroprostheses implies supplementary constraints. The whole control device has to be lightweight and worn by the subject. Hence, it is necessary first to minimize the equipment and second to propose efficient, lightweight solutions for visual scene analysis by the camera worn

by the subject. So it has to be a wearable device.

Real-time performance is also a mandatory requirement for our target application (Mick et al., 2019, 2021). As the fastest visuomotor response to a perturbation takes about 90 ms (Scott, 2016), and feedback delays of 100 ms or more are known to deteriorate the performance of online feedback control (Miall et Jackson, 2006). Therefore, computation time should remain as low as possible, and below 100 ms.

Field-Programmable Gate Array (FPGA) do fulfil those requirements, because they are lightweight, mobile, and also efficient accelerators for computer vision algorithms (Qasaimeh et al., 2019; Yu et al., 2020).

When speaking about the state-of-the-art (SOTA) family of computer vision algorithms allowing for object recognition and localisation, we have to remember that our environment is very much cluttered and various objects are present in it. In such real-world scenarios, we cannot reasonably hope that the SOTA object detection algorithm could perform well to identify the object-to-grasp. They are usually designed for much less cluttered scenes. External information is needed. The most natural way to introduce this information into prosthesis control consists in measuring user's intention by external tools and incorporating these measures into artificial vision approach to recognize and localize the object to grasp in the visual scene. A natural way to do it consists in measuring visual intention of the amputee with an eye-tracker (Mick et al., 2021). In this case, the gaze fixations on the images of a video recorded with scene view camera could guide the whole recognition system towards the object to recognize and localize.

Nevertheless, as shown in de San Roman et al. (2017), this information is noisy. First of all, when the person realises an initial visual search for object, gaze fixations are scattered in the scene, then some highly contracted visual distractors can disturb the subject in his visual search for the object. Hence, these information has to be

filtered from the past (initial observation) - to the current moment, estimating the gaze fixation on the object-to-grasp.

Filtering of gaze points with motion compensation from the past to the present frame is needed. To do this, the Scale-Invariant Feature Transform (SIFT) (Lowe, 2004) keypoints can be extracted in the images to compute the global motion model after matching of keypoints, to collect all gaze points in the current frame. This redundant information can then be used to estimate a smooth position of a gaze fixation in the current frame. Hence, in our work, we study and propose a solution for SIFT detector implementation on FPGA.

The most efficient solution for object recognition and localisation of it in the frames of video scene are convolutional neural network (CNN)s or recently proposed transformers such as Deformable Transformer for End-to-End Object detection (DETR) (Zhu et al., 2021). In our case, for object detection and localization we adapt and implement in hybrid FPGA - embedded CPU solution the gaze-driven DNN approach, previously developed in software in González-Díaz et al. (2019).

We present our work in the following chapters of the manuscript.

Chapter 2 contains the state-of-the-art (SOTA) in our problem, namely the Neuroprosthesis control, the object recognition and the FPGA implementation of visual scenes analysis methods.

Chapter 3 is devoted to the analysis of the algorithm developed in LaBRI for Object-to-grasp recognition (González-Díaz et al., 2019; Buzási, 2018; Poursanidis et al., 2020), which we have adapted for FPGA implementation (Fejér et al., 2019, 2021a,b, 2022). We supply the complexity analysis in terms of computational time for the hybridization choices.

Chapter 4 presents our solution for SIFT characteristic point detection (Lowe, 2004) on FPGA. We first detail the regular SIFT algorithm and its software im-

plementation reported by Hess (2010). Then we analyse the constraints for its implementation on FPGA and possible available solutions. Finally, we present our solution of accelerated SIFT approach adapted for FPGA implementation (Fejér et al., 2021a).

Chapter 5 contains the preprocessing steps of the Gaze-Driven CNN(González-Díaz et al., 2019). The preprocessing process has two steps: the gaze point alignment and the gaze point noise reduction. The gaze point alignment step contains the SIFT point extraction, the keypoint matching with the FLANN matcher (Muja et Lowe, 2009), and the homography estimation. The gaze point noise reduction has two steps: clustering the gaze points and elimination of the outliers, and the Kernel Density Estimation (KDE) (Pedregosa et al., 2011) estimation of the smoothed position of the gaze point in the current frame.

Chapter 6 presents the Gaze-Driven CNN originally proposed for computation on CPU (González-Díaz et al., 2019) and its FPGA optimized implementation (Fejér et al., 2022). The Gaze-Driven CNN FPGA optimized version (Fejér et al., 2022) is built in the following modules: the ResNet50 (He et al., 2016) which extracts the features from a frame, the Reduction Layer to change the size of the input layer of further network to a smaller one, the Faster R-CNN (Girshick, 2015) to estimate the grasped object location, and the Multiple Instance Learning (Amores, 2013) to predict the grasped object type.

Chapter 7 provides a conclusion of this research and outlines its perspectives.

Chapter 2

State-of-the-Art in visual scene analysis for neuroprostheses controls and FPGA implementation

2.1 Introduction

The neuroprosthetic arm is a great tool for helping disabled people who lost their arm due to a disease, an accident, or war injury. However, those robotic arms could be expensive and uncomfortable for the wearer. Nowadays, an intensive research is being conducted on how to achieve an affordable and comfortable, wearable robotic prosthetic arms.

In this Chapter 2 we first review the available methods for controlling a neuroprosthetic arm such as myoelectric control with elements of vision. Then we present an overview of the state-of-the-art for object recognition in visual scenes targeted to our application. Finally, we analyse the FPGA-based implementations of visual scene analysis methods as our goal is to propose such methods for prostheses control on hybrid architectures.

The researchers in assistive robotics and re-habilitation medicine state such as

Parker et al. (2006) that the myoelectric control only has its limitations. First of all the information measured as electrical signals of muscles is noisy and difficult to interpret. Second, the remaining muscles are not necessary the muscles which should control the missing joints. Some solutions have been proposed studying the co-contraction mechanisms in Parker et al. (2006), but this control remains quite sequential and unnatural. The behaviour pattern is difficult to learn by the amputee. Finally, the neuroprostheses with myoelectrical control could be invasive. Some of them require a surgery called "osseointegration" whereas the other surgery is the “targeted muscle re-innervation” (Farina et al., 2021). The latter improves the myoelectric control as it allows for activation of the nerve which controlled the missing muscles. But even in the case of myoelectric control of prostheses, with surface myo-sensors on the remaining part of the arm, that is without the surgery, they are not very comfortable for the amputees. They also do not ensure a perfect recording of myoelectric signals to control the artificial arm (Mereu et al., 2021).

The virtual reality (VR) (Karrenbach et al., 2022; Mick et al., 2021) and augmented reality (AR) (Shi et al., 2022b,a) have been already used for training amputees for neuroprosthesis control. The solutions based on deep learning (Karrenbach et al., 2022) or the regular feature extraction algorithms (Krausz et al., 2020) have been proposed. The AR can be used in the future for the prosthesis control, helping the amputee to select a good movement pattern to grasp an object. The AR or mixed vision-perception and myoelectrical control require understanding of the natural visual scene the amputee faces and, specifically, recognition and localization in it of the object that the person wishes to grasp;

In our case, it is important to know the type of the object and the location of the object in an image. The type is important because every object has a different shape, which requires a different grasp type. The location and distance are also important

for servoing the prosthetic arm. There are several methods for object recognition based on deep learning. For feature extracting there are VGGnet (Simonyan et Zisserman, 2015), Mobilnet (Howard et al., 2017), or ResNet (He et al., 2016). Faster R-CNN Ren et al. (2017) can use those feature extractors to recognize the object. There are also reviewed other object recognition methods such as Single-Stage Object Detection (SSD) (Liu et al., 2016) and YOLO (Redmon et al., 2016).

The developed device has to be wearable. This means that it has to be lightweight, and the device has to be portable, which means that it has to have low power consumption. FPGA has fulfilled those requirements, and it is also a powerful platform because there are methods to accelerate the algorithm. To accelerate a deep learning algorithm for a FPGA different tools are available: Vitis AI (Kathail, 2020), Apache TVM VTA (Moreau et al., 2019), Brevitas (Pappalardo, 2021), and FINN (Umuroglu et al., 2017). In the last part of Chapter 2 the different FPGA deep learning accelerators are reviewed and compared to each other.

2.2 Neuroprosthesis control based on visual and gaze information

Neuroprostethis control with visual and gaze information is a quickly developing research area. In this section, we will give a brief overview of the SOTA approaches. Karrenbach et al. (2022) implemented a data-driven predictive control strategy in object grasping tasks performed in virtual reality for wrist prediction to improve the performance of basic prostheses. Their method has two phases: in the first phase, data was captured for pregrasp hand poses, stopping at the point of contact to collect only relevant data for training. A set of eye-tracked gaze and hand kinematic data from a subject during an object reaching task in a virtual environment was

generated. The generated dataset is used during their CNN training. In the second phase, they implemented a user study with a full pick-and-place task to emulate a real use case and investigate the amount of compensation and effort that the user required in virtual reality. The evaluation was that the wrist prediction model was implemented in a virtual prosthesis and compared to a wrist-locked prosthesis. This method leads to a decrease in compensatory movement in the shoulder, as well as to a decrease in task completion time.

Mouchoux et al. (2021) developed a man-machine interface that endows a myoelectric prosthesis (MYO) with artificial perception, estimation of user intention, and intelligent control (MYO–PACE) to continuously support the user with automation while preparing the prosthesis for grasping. The Creative SR300 camera is on the glasses, which provides the camera looking the same scene as the user. It simultaneously acquires colour and depth (RGB-D) images at a resolution of 1920 $\times$ 1080 pixels and 640 $\times$ 480 pixels. Their method uses sEMG signals and Creative SR300 frames and depths to predict the right movements of the prosthetic arm and wrist. Mouchoux et al. (2021) system integrates a classification-based myoelectric control (MYO; implements the Linear Discriminant Analysis algorithm (LDA) (Englehart et Hudgins, 2003)) with a novel interface for comprehensive artificial exterior and proprioception and autonomous adaptive prosthesis control (PACE). Mouchoux et al. (2021) demonstrate that the implementation of advanced perception, context interpretation, and autonomous decision-making into active prostheses improves control dexterity. Moreover, it also effectively supports the user by speeding up the pre-shaping phase of the movement and decreasing muscle use.

Fukuda et al. (2021) proposed a novel control scheme for a vision-based prosthetic hand, which fuses bimodal information to achieve human-like hand movements. Their methods combine the sEMG signals and the visual information of the object

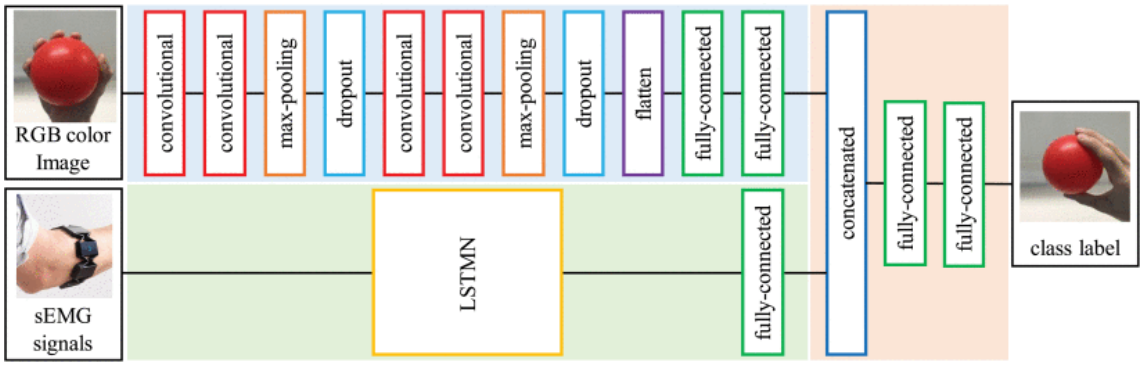


Figure 2.1: The bimodal information architecture (Fukuda et al., 2021).

to achieve control. It uses a deep convolutional network for servoing. In both the training and the recognition phase, the network uses the sEMG signals and the visual information of the object, called bimodal information. Fukuda et al. (2021) deep learning network has 3 subnetworks as a CNN for the frame input and an LSTMN for the sEMG signals and those are concatenated in the connection network and the result of this network is a class label. They assume only 1 object exists in a frame during the measurements. The camera (Microsoft LifeCam Studio for Business) is mounted on the prosthetic arm.

Shi et al. (2022b) designed a prosthetic arm control system based on eye-tracking. Their system is called i-MYO, and it uses EMG signals and the AR information from Microsoft HoloLens 2. The Microsoft HoloLens 2 is an AR helmet. The user wears this AR helmet and can switch the type of grasp with his eyes. It has 6 different grasp types: Cylindrical, Spherical, Tripod, Pinch, Lateral, and Hook. The user has to look at the selected grasp type for 200 ms. This grasp-type switching is called i-GSI (Shi et al., 2022a). After the selection, the system is used the EMG signals to get the intention of the user to control the prosthetic hand opening or closing.

Krausz et al. (2020) made a system for controlling a prosthetic arm which fused EMG and gaze data predict the desired end-point for full arm prosthesis, which could drive the forward motion of individual joints. The image processing algorithms

have been used to resolve the concurrent head motion in gaze tracking. The image is processed from the front-facing camera to project the gaze position into a global 2D space (using the first recorded frame as a reference). The ORB (Rublee et al., 2011) features and the BRIEF (Calonder et al., 2010) descriptors are extracted in each image, and the extracted descriptors are matched with the FLANN-based matcher (Muja et Lowe, 2009) in all consecutive frame pairs. Then RANSAC (Fischler et Bolles, 1981) was used to estimate the homography transformation matrix to project each pixel from the head-frame to the global frame, and produce gaze coordinates independent of head motions and solely dependent on the distance between targets in the global frame. This gaze fixations result has been fused with the EMG data using a Kalman filter (Kalman, 1960).

Mick et al. (2021) proposed a system which fuses the myoelectric and kinematic information with the gaze information. The gaze tracked information is good to get contextual information about the target's location and orientation. To do that they are using computer vision algorithm such as Neural network. Mick et al. (2021) also used virtual reality. They are created in a virtual environment which was scaled to match the real-world dimensions. The subjects used HTC Vive Pro virtual reality headset and HTC Vive Tracker motion tracker. The subject's goal is to grasp and move a bottle.

The problem with VR (Karrenbach et al., 2022; Mick et al., 2021)) and AR (Shi et al., 2022b,a) based solution is that it could cause visual fatigue and motion sickness as mentioned by Chang et al. (2020); Park et al. (2014); Lambooij et al. (2009). The symptoms include but are not limited to eye fatigue, disorientation, and nausea. Today is a big research area to solve the caused problems of the VR and AR (Kramida, 2016), however it is still an open question. So VR and AR are not good for real clinical research which is available in the outside world, but

it is good for getting proper data for the training (Mick et al., 2021). In the real-world environment, the visual scene analysis tools have to be sufficiently advanced to recognize and localize, in a cluttered environment, the object the prosthesis wearer needs to grasp. In the follow-up of this chapter we will review popular methods from the state-of-the-art (SOTA) for object recognition.

2.3 Object recognition in visual scenes

In recent years, in the field of computer vision, the most popular algorithms for object detection are deep convolutional neural network, such as Fast Region-based Convolutional Neural Network (R-CNN) (Ren et al., 2017), You Only Look Once (YOLO) (Redmon et al., 2016), and Single-Stage Object Detection (SSD) (Liu et al., 2016). These detectors are based on deep Residual Network (ResNet) (He et al., 2016), Visual Geometry Group Net (VGGnet) (Simonyan et Zisserman, 2015), Alexnet (Krizhevsky et al., 2017), MobileNet (Howard et al., 2017), GoogleNet (Szegedy et al., 2015) and on older Alexnet (Krizhevsky et al., 2017).

ResNet (He et al., 2016) was proposed by He et al. and uses residual blocks, which are illustrated in Figure 2.2.

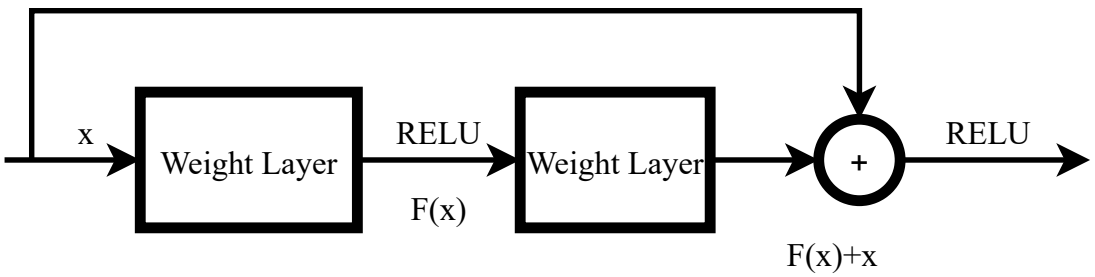


Figure 2.2: Example of the residual block in the ResNet.

Its principle is based on the optimizing a residual mapping instead of direct mapping.

Let us denote the desired underlying mapping as $H(x)$. Then the new mapping $F(x)$ is introduced as

$$F(x) := H(x) - x$$

where we let the non-linear stacked layer fit $F(x)$. The original mapping is recast into $F(x) + x$. It is easier to optimize the residual mapping than to optimize the original mapping. $F(x) + x$ can be realized by feedforward neural networks with shortcut connections, as illustrated in Figure 2.2. Shortcut connections can skip one or more layers. In ResNet (He et al., 2016), the outputs of the shortcut connections are simply added to the outputs of the stacked layer.

The computational cost of ResNet (He et al., 2016) is high, which makes real-time implementation difficult. However, there are methods that can accelerate the computational speed.

VGGNet (Simonyan et Zisserman, 2015) is a simple deep convolutional neural network, where depth refers to the number of layers. The VGG-16 consists of 13 convolutional layers and 3 fully connected layers. The convolutional layers are simple because they use only 3×3 filters and pooling layers. This architecture has become popular in image classification problems.

Faster R-CNN was proposed by Ren et al. (2017). This architecture has gained popularity among object detection algorithms. Faster R-CNN (Ren et al., 2017) is composed of the following four parts:

- feature extraction module; this can be a VGGnet (Simonyan et Zisserman, 2015), Mobilenet (Howard et al., 2017), or ResNet (He et al., 2016);
- region proposal module to generate the bounding boxes around the object;
- classification layer to detect the class of the object—for example, cat, dog, etc.;

- regression layer to make the prediction more precise.

The computational speed of the network depends on the feature extraction module and the size of the region proposal module.

Both SSD (Liu et al., 2016) and YOLO (Redmon et al., 2016) are single-stage detectors. They are significantly faster than two-stage detectors (region-based methods), such as Faster R-CNN (Ren et al., 2017). However, in cases when the objects have not so much variability, neither interclass nor intraclass Faster R-CNN (Ren et al., 2017) is a well-suited network. In our problem, we are interested in naturally cluttered home environments, where the subject intends to grasp an object, such as in kitchens. The vision analysis system we propose has to be designed to recognise objects to grasp in the video, similar to the Grasping In The Wild (GITW) dataset (LaBRI, 2016). This dataset was recorded in natural environments by several healthy volunteers and made publicly available on the CNRS NAKALA platform. The objects here, seen from the glass-mounted camera, are quite small. Their surface merely represents 10% of the whole video frame. For more information on this dataset, see Chapter 3.3.2. Hence, Faster R-CNN (Ren et al., 2017) is a better choice than the SSD (Liu et al., 2016) and YOLO (Redmon et al., 2016). This is due to the fact that Faster R-CNN (Ren et al., 2017) achieves higher mean average precision (mAP) than them, as reported by Huang et al. (2017) for small objects.

The original Faster R-CNN (Ren et al., 2017) uses VGGnet (Simonyan et Zisserman, 2015) as a feature extractor. However, the mAP is higher when ResNet (He et al., 2016) is used as a backbone (Redmon et Farhadi, 2016). When the object is small, the mAP of the backbone with ResNet (He et al., 2016) is higher than the backbone with MobileNet (Howard et al., 2017), as reported in Huang et al. (2017).

2.4 FPGA implementation of visual scenes analysis methods

There are several possible ways to accelerate an algorithm, as described in Fejér et al. (2019). In our case, the FPGA was chosen in the interest of developing a lightweight and portable device (Fejér et al., 2021b). In the first part of the section, the different CNN accelerations on the FPGA are discussed. These approaches have been implemented as tools available today to accelerate a network on different FPGAs.

The second part of this section is about the acceleration of the SIFT (Lowe, 2004) algorithm on different hardware architectures. The main parameters of various architectures will be discussed from the point of view of the wearable device.

2.4.1 CNN acceleration on FPGA

If speaking of one of possible realistic scenarios for a prosthetic arm control, the DNN for object detection and recognition can be trained off-line. The real-time constraints are imposed only at the inference stage. Neural network inference can be very efficiently accelerated on Field-Programmable Gate Array (FPGA). The most important frameworks and development environments are Vitis AI (Kathail, 2020), Apache TVM Versatile Tensor Accelerator (VTA) (Moreau et al., 2019), Brevitas (Pappalardo, 2021), and FINN (Umuroglu et al., 2017).

Due to large computing and memory bandwidth requirements, deep neural networks are trained on high-performance workstations, computing clusters, or GPUs using floating-point numbers. The memory access pattern of the inference step of a trained network is different, offering more data reuse and requiring smaller mem-

ory bandwidth. It makes FPGAs a versatile platform for acceleration. Computing with floating-point numbers is a resource-intensive process for FPGA in terms of digital signal processing (DSP) slices and logic resource usage. Memory bandwidth, required to load 32-bit floating point state values and weights, can be still high compared to the capabilities of low-power FPGA devices. Additionally, a significant amount of memory is required for buffering state values and partial results in the on-chip memory of the FPGA. One possible solution would consist of using the industry-standard bfloat, 16-bit, floating-point representation, which can improve the inference speed of an FPGA. Observations show (Umuroglu et al., 2017) that the value of weights, state values, and partial results during the computation usually fall in a relatively small range and the 8-bit exponent range of the bfloat type is practically never used. If the range of values during the computation is known in advance, then fixed-point numbers can be used. One of the major application areas of FPGAs is signal processing; therefore, DSP slices are designed for fast, fixed-point Multiply-ACcumulate (MAC) or Multiply-ADD (MADD) operations, which can be utilized during neural network inference.

Converting a neural network model trained with floating-point numbers to a fixed-point FPGA-based implementation usually requires an additional step, called quantization. Here, a small training set is used to determine the fixed-point weights and optimize the position of the radix point in each stage of the computation. The common bit width for quantization is 16 or 8 bits, where the accuracy of the network is slightly reduced. In some cases, even a binary representation is possible (Umuroglu et al., 2017), eliminating all multiplications from the computation, which makes FPGA implementation very efficient while the accuracy is decreased slightly.

For latency-sensitive applications, this fixed-point model can be implemented on a streaming architecture, such as FINN (Umuroglu et al., 2017), where layers of

the network are connected directly on the FPGA. Using this structure, loading and storing state values can be avoided. In an ideal case, when the number of weights is small enough, they can be stored in the on-chip memories, further reducing the memory bandwidth requirements of the system. This also results in lower dissipated power due to the high energy requirement of off-chip data movement. Another approach used in Vitis AI (Kathail, 2020) and Apache TVM VTA (Moreau et al., 2019) is to divide the computation into a series of matrix-matrix multiplications and create a customized ISA (Instruction Set Architecture) to execute these operations efficiently. The resulting system might have higher memory bandwidth requirements and longer latency, but can be easily reprogrammed to infer a different network during different steps of an image processing application.

Apache TVM VTA (Moreau et al., 2019) is an open, generic, and customizable deep learning accelerator with a complete TVM-based compiler stack. It is an end-to-end hardware-software deep learning system stack that combines TVM and VTA. It contains the hardware design drivers, a Just-In-Time (JIT) runtime, and an optimizing compiler stack based on TVM.

The main advantages of quantization are reduced circuit complexity, efficient use of dedicated hardware resources, reduced on-chip memory requirements, reduced off-chip memory bandwidth, and smaller power dissipation. Therefore, for a lightweight body-worn device, Vitis AI (Kathail, 2020) is a good choice, because it can accelerate the network with minimal loss of accuracy.

2.4.2 SIFT acceleration on FPGA

Detecting SIFT key-points is a part of the whole vision assistance for prosthesis control and also is one of the bottlenecks in computational speed. Computing power is a multi-parameter vector: any algorithm solving a problem will have a speed-

power-area-bandwidth-accuracy metric. In the follow-up of this section, a review of available solutions is being presented based on this multi-criteria point of view.

First of all, detecting keypoints in images for matching them in mosaicking or motion compensation is a very well studied problem. There are several keypoint detectors existing in computer vision such as Oriented FAST and Rotated BRIEF (ORB) (Rublee et al., 2011), Speeded-Up Robust Features (SURF) (Bay et al., 2008), and the most widely used is SIFT (Lowe, 2004). The previously mentioned detectors differ in the following way, one of which is the technique by which keypoints are extracted and also by the process the descriptors are computed and matched. ORB uses two main steps: an oriented Features computation from Accelerated Segments Test (FAST) (Rosten et Drummond, 2006) which extracts the keypoints location and orientations of the keypoint combined with Binary Robust Independent Elementary Feature (BRIEF) (Calonder et al., 2010). The latter calculates the binary descriptors around extracted points. The SURF (Bay et al., 2008) uses integral images and box filtering to extract the keypoints and descriptors computation. Both SURF (Bay et al., 2008) and ORB (Rublee et al., 2011) require less computational time than SIFT (Lowe, 2004). All these detectors were proposed to accelerate and simplify the original SIFT detector proposed in (Lowe, 2004). Nevertheless, we have chosen the SIFT algorithm to implement on FPGA because Karami et al. (2017) showed that in different kinds of transformations and deformations such as scaling, rotation, noise, fish-eye distortion, and shearing SIFT outperformed other methods in precision.

SIFT is a widely used and implemented keypoint detector. There are CPU (Bradski, 2000; Hess, 2010), GPU (Li et al., 2017; Björkman et al., 2014) and FPGA implementations (Ginés et al., 2020; Pablo et al., 2018; Vourvoulakis et al., 2017, 2016; Chang et al., 2013; Shao et al., 2015). The OpenCV SIFT library (Bradski,

2000) and the OpenSIFT (Hess, 2010) are popular frameworks for SIFT keypoint extraction and descriptor computation on CPUs. This is partially due to their flexibility, as the input image resolution can be modified easily. However, the runtimes of these CPU implementations are too slow for real-time image processing on lightweight devices.

Computation of SIFT can be accelerated by using GPUs, for example: CudaSIFT (Björkman et al., 2014) can process a 1280px × 960px image in 12.7 ms, i.e. 78.74 frames per second (fps) on the NVIDIA GeForce GTX 580 GPU. However, its power consumption is 244W, which is too high for a wearable application. HartSIFT (Li et al., 2017) can extract features within 3.14∼10.57ms (94.61∼318 fps) depending on the input image size on the NVIDIA GeForce GTX TITAN Black. The power consumption of this GPU is 250W, which is also very high for a portable device. da Costa Barreiros (2020) implemented a SIFT Lowe (2004) on GPUs. da Costa Barreiros (2020) tested the implemented SIFT on different GPUs like NVIDIA GTX 1060, NVIDIA GTX TITAN Black, and the embedded system NVIDIA Jetson TX2. The power consumption of the NVIDIA Jetson TX2 is 7.5W-15W, which is ideal for a portable device. da Costa Barreiros (2020) SIFT can process a 2560 × 1920px image 1142.56ms on NVIDIA Jetson TX2. It can compute the Scale-Space in 411.32ms, the Difference of Gaussians in 11.43ms, the extrema detection in 262.79ms, the orientation histogram in 68.55ms and the feature descriptor step in 388.47ms. So it can compute the KP extraction step in 685.54ms.

The most computationally intensive operation in SIFT keypoint extraction is the computation of the Gaussian pyramids, as it requires the multiplication of coefficients of Gaussian filters with scale-space images. For this step, an analog solution is developed by Rodríguez-Vázquez et al. (2009); Suárez et al. (2014) where the Gaussian pyramid is computed by an analog CMOS circuit and exhibits very low

dissipated power. One of the advantages of the inherent parallel processing is the high computational power of the analog VLSI implementations. However, the size of the array is small (Rodríguez-Vázquez et al., 2009; Toshiba, 2019; Suárez et al., 2014). In Suárez et al. (2014), the analog sensor/processor implementation has 88x60 processing elements, and each processing element has 4 photo-diodes. The computation unit is connected to the vision sensor unit of the camera. The vision sensor array has 176x120 pixels only and the system is implemented using a 0.18 μm CMOS technology. This solution is satisfactory from a computational point of view. However, it is not flexible with regard to increasing the video resolutions on wearable cameras. The resolution of the analog sensor is not sufficiently high for our application, since our method uses larger areas in video frames and it needs 480px $\times$ 480px image as input at least, if the resolution of video frames is HD or Full HD, which is a case of many commercial video cameras on glasses or on other wearable devices.

Several different SIFT implementations on FPGA are published, such as the system designed by Ginés et al. (2020), which is a simplified version of the algorithm. It is assumed that each feature point has two main orientations at most, the orientation histogram uses 8 bins instead of 36 bins. Thus, complexity is reduced, but the precision of orientations drops with regard to the original SIFT (Lowe, 2004). The system can process 640px $\times$ 480px sized input images at 99 fps processing speed on Xilinx Virtex-5, and it uses fixed-point representation.

Pablo et al. (2018) implemented the subpixel refinement stage of the SIFT algorithm. They used a ZedBoard (Xilinx Zynq7020) FPGA board and the rest of the algorithm was computed on the CPU.

Vourvoulakis et al. (2017, 2016) implemented an FPGA accelerated SIFT matching with RANSAC support. The architecture includes 1 octave and 4 scales. The

Gaussian kernel computation is optimized to reduce the logic and memory resource requirements on the FPGA. Size of the circuit is further reduced by using 14bit fixed-point numbers during the Gaussian filter computation. In order to evaluate the first derivatives of the Difference of Gaussian images, Prewitt mask has been used. This step is required to remove the keypoints on the edges. Their system was implemented using VHDL on an Intel DE2i-150 board and can process 640px $\times$ 480px input video at 81 fps.

Chang et al. (2013) proposed an architecture, where the Scale-space extrema are calculated on the FPGA and the rest of the algorithm on the CPU. The scale-space extrema detection uses a separable kernel, which requires reduced logic and memory resource usage on the FPGA. Their implementation runs in a Xilinx Virtex II Pro FPGA, with a configuration of three octaves and six scales, and with a 145 MHz clock frequency. An image of 320px $\times$ 240px is processed in 1.1 ms (900 fps).

Shao et al. (2015) implemented the SIFT algorithm on a Virtex-5 FPGA board, the input image has 292px $\times$ 520px resolution. The system can process images at 38 fps. They changed the Gaussian pyramid building, instead of filtering in parallel with different size filters they cascaded several smaller size Gaussian kernels. The design of the system is simplified but requires more memory resources.

There also exists an FPGA implementation which handles the descriptor matching part after the keypoint extraction (Daoud et al., 2020). The architecture is fully pipelined and uses a 16-bit fixed-point number representation.

Hence, the efficient SIFT implementation on FPGA still remains an open research question.

In Chapter 4.3, we propose an FPGA solution which is developed in a high-level language. Therefore, it is easier to modify some parameters such as the number of Gaussian layers compared to VHDL or Verilog-based SIFT implementations.

2.5 Comparing different hardware for an embedded system from computer vision point of view

For controlling prosthetic arm a wearable, lightweight device is mandatory. There are two ways to implement a wearable device. One possible solution to use an IoT device which contains one or more sensors and a small micro-controller for network communication, data recorded by the sensors being processed by a remote server. Another possibility is to design an embedded system where both data recording and processing is performed by the same place. Both of the solutions are good for creating a lightweight and portable device. We are decided to design an embedded system.

For creating an embedded system there are different hardware architectures such as Microcontrollers, Microprocessors (Raspberry Pi), Nvidia Jetson and FPGA SoC. All of them have a low power consumption however the Microcontrollers and Microprocessors do not have enough computing power.

Qasaimeh et al. (2019) compared ARM57 CPU, Jetson TX2 GPU, and ZCU102 FPGA to each other. They used publicly available computer vision and image processing libraries such as OpenCV, Nvidia VisionWorks and xfOpenCV, without adding any special platform specific code. Qasaimeh et al. (2019) experiments show that simple and easy-to-parallelize kernels perform on GPUs (1.1-3.2x energy/frame reduction). FPGAs outperform GPUs and CPUs (1.2-22.3x energy/frame reduction) in more complete vision pipelines. Thus, FPGAs perform better if the complexity of vision pipelines grows.

Yu et al. (2020) implemented an FPGA-based Overlay Processor for Lightweight convolutional neural network namely Light-OPU on Xilinx XC7K325T FPGA board. They compared different hardware such as Intel(R) Core(TM) i7-8700K CPU @

3.70GHz and Nvida Jetson TX2 GPU. Inferences on GPU use batch = 1 mode for latency dominating evaluation, with power consumption all averaged over 500 runs. Yu et al. (2020) present a comprehensive evaluation of eight lightweight CNNs such as MobileNetV1 (Howard et al., 2017), MobileNetV2 (Sandler et al., 2018), MobileNetV3 (Howard et al., 2019), SqueezeNet (Iandola et al., 2016), DenseNet (Iandola et al., 2014), Xception (Chollet, 2017) and ShuffleNet (Zhang et al., 2017). They also experimented with different kernel sizes (1×1, 3×3, 5×5, 7×7), strides (1×1, 2×2), layer types (Conventional-CONV, DW-CONV, group-CONV) and report these results. Irregular operations such as channel shuffle, residual addition and dense block concatenation are also included. Yu et al. (2020) solution performed 5.5× better latency and 3.0× better power efficiency compared with edge computing targeted GPU Jetson TX2.

Based on Qasaimeh et al. (2019); Yu et al. (2020) FPGAs compared to GPUs are powerful and more power-efficient so in our case, it is an ideal choice to create an FPGA-based embedded system.

2.6 Existing hybrid HW/SW solutions on FPGA

In the variety of proposed hybrid solutions today, there exist FPGA boards with system on chip (SoC) such as Zynq UltraScale+ MPSoC ZCU102 (Xilinx, 2019). The features of those boards are that a direct connection between an embedded CPU and the FPGA blocks is ensured. This is a good feature of such boards, as the implementation of a complex algorithm can be realized in a hybrid way : low-level signal processing operations necessary for image analysis algorithms can be implemented in FPGA blocks, while more sophisticated parts of them, which does not require purely hardware acceleration can be fulfilled on the embedded CPU.

Nowadays, the acceleration of computer vision algorithms is necessary not only in assistive robotics, but also in industrial robotics and UAV embedded video analysis Kóta et al. (2019), scene understanding for self-driving car guidance Han et Oruklu (2014). The implementation has to be lightweight and with a high computational power, as in our case.

Thus, Kóta et al. (2019) proposed a hybrid solution for a collision avoidance system based on visual detection, for Unmanned Aerial Vehicles (UAVs). The whole algorithmic chain of the solution comprises two parts:the preprocessing algorithms which crop the horizon, and the clouds, and a neural network, that identifies the approaching object. Some parts of image processing algorithm at the pre-processing step are accelerated on FPGA, such morphological operations (erosion, dilation), Gaussian blurring, and adaptive thresholding. The neural network and the contour search algorithm for obstacle avoidance run on the embedded processor.

Han et Oruklu (2014) developed a hybrid HW/SW system for real-time traffic sign detection and recognition. The hue calculation and the morphological filter are running on FPGA, while the filtering for good points and the other part of the decision is running on the embedded ARM processor. Their results show that the HW/SW solution is faster than the soft-core CPU implemented version.

Sun et al. (2018) designed a Zynq-7020 based system for multi-axis motion control and motor drive of robotic arms. The embedded CPU realizes advanced servoing algorithm, and accomplishes multi-axis trajectory planning, while the FPGA realizes the Multi-axis pipeline current loop controller. Their results show that controlling a robotic arm on FPGA is achievable.

Hence, Kóta et al. (2019); Han et Oruklu (2014) have shown that for computer vision and image processing problem a *hybrid* HW/SW embedded system could be a good solution for creating a portable, low-power consumption and wearable device.

Sun et al. (2018) demonstrated that the hybrid FPGA embedded system is also good for just controlling the motors of the robotic arm.

2.7 Conclusion

Hence, in this chapter we have presented a focused SOA analysis for our target task: real-time implementation of the object-to-grasp detection and localization on a light-weight wearable device from ego-video and gaze fixations information in view of prosthetic arm control.

Summarising different research contributions, we can state the following.

Mouchoux et al. (2021) used gaze tracking devices to control a robotic arm. Fukuda et al. (2021) are also controlling a robotic arm with visual guidance and they are using deep learning methods. Therefore controlling a robotic arm with visual guidance is possible.

The FPGA-based solution is ideal for controlling a robotic arm based on visual information, because it is a good accelerator for computer vision and image processing (Qasaimeh et al., 2019; Yu et al., 2020). FPGA also outperformed GPU in power efficiency and latency (Yu et al., 2020). The FPGA has direct I/O access which GPU does not have.

Kóta et al. (2019); Han et Oruklu (2014) showed that a hybrid embedded system is feasible for implementing computer vision and image processing algorithms and accelerating them for a real-time processing. They also showed that those embedded systems are energy efficient and wearable, which is necessary in our case to control a robotic arm. Sun et al. (2018) showed that a hybrid HW/SW system is convenient for controlling the robotic arm motors. Hence, the whole end-to-end implementation for visual control of a robotic prosthetic arm can be done on the same wearable

device.

Coming to the methodological tools of the whole algorithmic chain to implement, we first have compared different keypoint extractors such as SURF (Bay et al., 2008), ORB (Rublee et al., 2011) and SIFT (Lowe, 2004). SIFT has been chosen to implement to an FPGA, because Karami et al. (2017) showed that in different kinds of transformations and deformations such as scaling, rotation, noise, fish-eye distortion, and shearing SIFT outperformed other methods in precision.

For the choice of object-to-grasp recognition framework, we found our decision on the real-world video corresponding to the ecological situation of the prostheses wearers, namely the GITW LaBRI (2016) dataset. In case of the camera-on-glasses setting the objects are small, their surface merely represents 10% of the whole video frame. In that case, Faster R-CNN (Ren et al., 2017) is a better choice than the popular SSD (Liu et al., 2016) and YOLO (Redmon et al., 2016) due the fact that Faster R-CNN (Ren et al., 2017) achieves higher mAP as it was reported by Huang et al. (2017) for small objects.

The original Faster R-CNN (Ren et al., 2017) implementation uses VGGnet (Simonyan et Zisserman, 2015) as a feature extractor. Redmon et Farhadi (2016) showed that the mAP is higher when ResNet (He et al., 2016) is used as a backbone. Huang et al. (2017) reported that when the objects are small the mAP is higher when ResNet (He et al., 2016) has been used as backbone compare to MobileNet (Howard et al., 2017).

Finally, we analysed different CNN accelerators on FPGA. And we come to the conclusion, that Vitis AI (Kathail, 2020) is a good choice for accelerating a CNN, because it quantizes the network. The advantages of quantization are reduced circuit complexity, efficient use of dedicated hardware resources, reduced on-chip memory requirements, reduced off-chip memory bandwidth, and smaller power dissipation.

Vitis AI (Kathail, 2020) can accelerate a CNN with minimal loss of accuracy. Also the Vitis AI is suitable for creating a wearable device.

In the next Chapter 3, we will analyse the full algorithmic chain of object to grasp recognition in view of FPGA based implementation.

Chapter 3

Analysis of object-to-grasp recognition in view of FPGA based implementation

3.1 Introduction

The vision analysis part is the most critical in the entire chain of prosthetic arm control. In fact, errors in visual recognition and object localization would yield the wrong prosthetic arm servoing and would require corrective actions from the prosthesis wearer. Obviously, none of the automatic systems gives an ideal solution of 100% accuracy, one of the best reported results are as low as of MSE = 52.14 $\pm$ 7.72 cm^2 accuracy (Krausz et al., 2020), but corrective actions, induced by wrong recognition and localization, have to be as rare as possible. Therefore, regardless of the algorithmic complexity of the approach, the first goal of vision-guided prosthesis systems is to ensure the best possible success rate. However, Ortiz-Catalan et al. (2015) has demonstrated that quantitative evaluation metrics used to assess a control's correctness under laboratory circumstances are poor indicators of actual clinical results. Therefore, we cannot reduce the performance, but need to accelerate

the computations in a lightweight framework of a wearable device.

The complexity of visual recognition algorithms today, in particular object recognition, is such that we can not reasonably hope to implement all the processing chain on FPGAs. Filtering, or other low level image processing algorithms (Zarándy et al., 2016) can be easily implemented as we reviewed it in Chapter 2. Today many components to build object recognition systems have been already made available in FPGAs implementation such as CNNs (Fan et al., 2018). Therefore, the goal is not to implement the whole approach for object recognition in FPGAs, but to propose a flexible solution which would allow for:

- fast adaptation to new achievements in object recognition;
- be compatible with real time for prosthetic arm control;
- ensure low dissipation of power for wearable devices;

Therefore, we have to identify the most critical components of the algorithm in terms of

- computational power required,
- availability of component implementations
- inter changeability of components

In this Chapter 3 we analyze the full chain of the system, we present the GITW (LaBRI, 2016) dataset which was recorded for the object grasping scenario, the target FPGA board. We finally analyze the computational complexity of the system for object-to-grasp recognition in view of its hybrid implementation.

3.2 Visual servoing of upper-limb prosthesis scenario

In our vision servoing scenario, the prosthesis wearer is supposed to wear glasses with a scene camera and an eye tracking device. Hence, to elaborate our solution, we have used a set of video pre-recorded with such a setting.

The Tobii Pro Glasses 2 eye tracking system (AB, 2016) has been used to record the videos and the gaze points during the experiments. The system comprises a lightweight Tobii Pro Glasses Head Unit, a wearable Tobii Pro Glasses Recording Unit and Tobii Glasses Controller (running on Windows 7, 8 or newer operating system) or Tobii Pro Glasses 2 API which is running on any devices. The Tobii Pro Glasses Recording unit can be worn as a glass as shown in Figure 3.1.

Figure 3.1: Example of the Tobii 2 glass camera usage.

Before starting an experiment, the system must be calibrated for each participant. During the calibration process, the user should look at the calibration card held in front of him for a few seconds. When the system is calibrated, the recording can be started.

The Tobii Pro Glasses Recording Unit can record an egocentric video 1920px × 1080px size with 25fps. It can also record the user's gaze point location in the video frame and the distance between the glass and the foveated object with millimeter precision. The gaze point data are stored in JSON file format, and it can be accessible via the API or with the Tobii Glasses Controller.

The Tobii Pro Glasses Recording Unit has four API interfaces, the POST API, the REST API, the Livestream API, the Discovery API.

- POST API: The API is stored on the SD card and contains all scene camera and gaze data stored during recording and calibration.
- REST API: This API is used to control the Tobii Pro Glasses Recording Unit, e.g. to create projects, start and stop calibrations and recordings, but it can also be used to retrieve Tobii Pro Glasses Recording Unit status and information of the Tobii Pro Glasses Recording Unit and its head unit.
- Livestream API: This API can be used to get live data and video in real time.
- Discovery API: This API can be used to discover a Tobii Pro Glasses Recording Unit over the network.

3.3 Object-to-grasp recognition approach

The vision analysis part, which is the most critical in the whole chain of prosthesis servoing, is presented in Figure 3.2. The underlying hypothesis for the function-

ing of vision-guided neuroprostheses is that the upper limb amputees wearing the neuroprostheses are first looking at the object they wish to grasp.

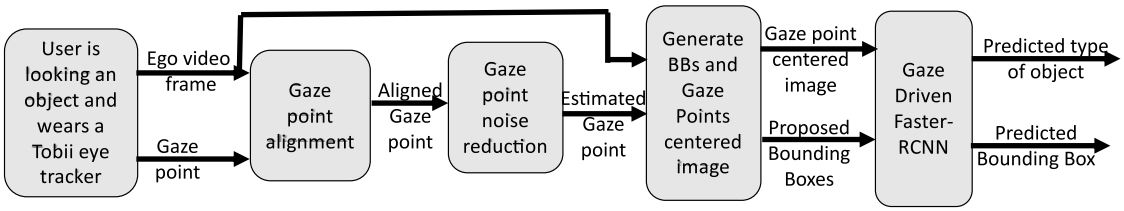


Figure 3.2: The prosthetic arm visually guided system.

The subject is wearing a Tobii glasses device, which acquires an ego-visual scene and records gaze fixations of the subject in their coordinate system—see the left-most block in Figure 3.2. The recorded gaze fixations allow for roughly localizing the object of interest in video frames.

Nevertheless, visual saccades to the distractors in a visual scene, microsaccades, and initial scene exploration before the subject finds the object make these measurements noisy. Hence, two blocks of the system—gaze point alignment and gaze point noise reduction—serve to estimate the position of the gaze fixation on the object in the current ego-video frame.

The gaze-point alignment module aims at estimating and compensating for the ego-motion between the past frames and the current frame. For more details, see Section 5.2.

The goal of the gaze point noise reduction module is to reduce the noise in the current frame. This noise can be a head motion, or a product of the user being distracted and looking at another object for a moment. For more details, see Section 5.3.

Then, the video frame is cropped around the estimated gaze point to limit the area of the object search. Finally, different object proposals bounding boxes (BBs) at different scales are generated around the point for object localization. The gaze

point-centred image and the set of BB coordinates are then submitted to the gaze-driven CNN—see the right-most block in Figure 3.2.

The gaze-driven CNN (González-Díaz et al., 2019) is pre-trained on the taxonomy of objects to detect. It outputs the best score for the object class and the best-scored bounding box. When the object is localized in a video frame, the 3D position of it for prosthesis servoing can be estimated from eye tracker depth measures of gaze fixation and the coordinates of the centre of the best-scored bounding box.

The resolution of the Tobii first-person view camera is full HD (1920 × 1080 px), with a frame rate of 25 frames per second (fps). More information about Tobii can be found here in Chapter 3.2

The real-time requirement for the system in our case means that each processing step of the localization of the object of interest in the glasses-mounted camera in a current video frame has to be lower than 40 ms (the video acquisition rate), and the latency of the whole system should be lower than 100 ms to leave the place for mechanical servoing of the prosthetic arm (Mick et al., 2019).

In this work, we do not consider depth estimation, which is a simple regression from eye tracker gaze fixation measures—our focus is on object detection. In the following passages, we present each system block in detail.

3.3.1 Object recognition scenario

González-Díaz et al. (2019) proposed a Gaze-Driven Faster R-CNN, which can predict an object type and location in an image. Their input is the Tobii (AB, 2016) eyetracker camera frame and the recorded gaze point, more information about Tobii camera can be found here in Chapter 3.2.

González-Díaz et al. (2019) system has a preprocessing part, which is built upon two different modules, see the upper part of Figure 3.3: the geometric alignment and

the gaze point noise reduction. The geometric alignment module aims at estimating and at compensating the ego-motion between the past frames and the current frame. This step is necessary to project all collected gaze points into the current frame. These gaze points will be used to estimate a smoothed position of the gaze point in the current frame, as the real gaze point coordinates can be noisy due to the distractors and micro-saccades. Our solution is using the same preprocessing steps. For more information, see Chapter 5.2.

The gaze point noise reduction aims at reducing the noise caused by a head motion, or produced by the user being distracted and looking at another object for a moment. González-Díaz et al. (2019) used the Kernel Density Estimation (KDE) algorithm to do that. The input was the geometrically aligned gaze points and the KDE searched in the whole image. The output is the estimated fixated gaze point. For more information about KDE, see Chapter 5.3.2.

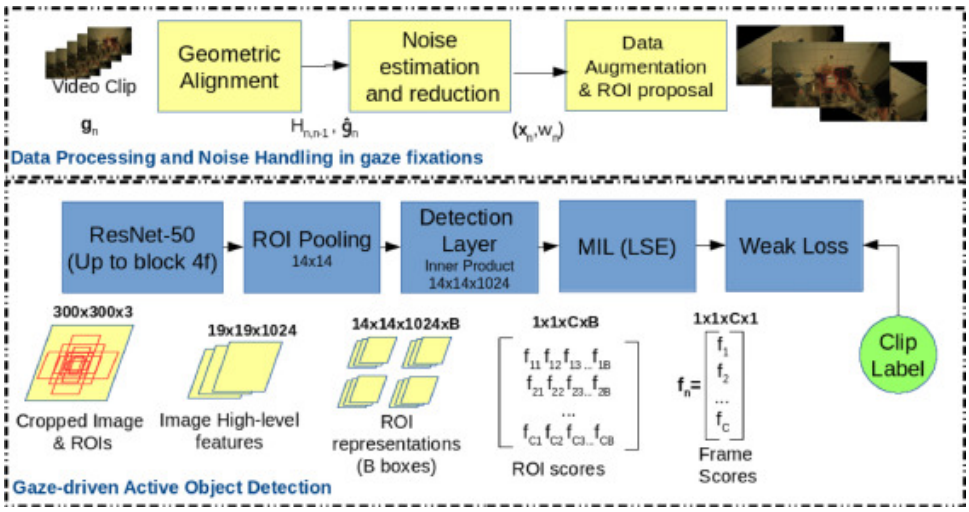


Figure 3.3: Gaze-driven CNN for object recognition. Upper part: Preprocessing pipeline. Lower part: the Gaze Driven CNN on object proposals (González-Díaz et al., 2019).

To localize the object more precisely, González-Díaz et al. (2019) proposed to generate a certain number of object proposals, such as bounding boxes of different scales and aspect ratios around the estimated gaze point in the current frame. In

the original work González-Díaz et al. (2019) the number of different BBs was 9. We keep it in our hybrid implementation.

So the Gaze-Driven Faster R-CNN (González-Díaz et al., 2019) inputs are

- the 9 bounding boxes around the estimated gaze point;
- the current frame;

As feature extractor from the current frame, we use ResNet (He et al., 2016) subnet. Note that González-Díaz et al. (2019) once the gaze fixation is estimated, there is no need to search for the object in the whole frame. In González-Díaz et al. (2019) an input crop of 300×300px (centered on the estimated gaze point) is performed and submitted to ResNet. The resulting generated feature tensor is of size 19×19×2048, with a reduced spatial dimension (19×19) and an extended set of 2048 high-level feature channels.

Then the individual representation is generated for each considered candidate region of interest (ROI)/bounding box (BB). The Multi-scale ROI Align pooling (Lin et al., 2016) was used to generate the individual representations, and it produces a set of bounding boxes 14x14x2048 tensors associated to each of the candidate ROIs as shown in Figure 3.3. When an independent representation of each ROI is computed, then the next step is to calculate the detection score for each object category with the detection layer. The detection layer is a fully-connected layer that transforms the 14x14x2048 tensor into a 1x1x17x9 (1x1xCxB) length tensor where C is the class number and B is the bounding boxs around the object of interest. The class number is 17 as the taxonomy comprises 16 object classes plus the background. In the final step the vectors of all ROIs, are concatenated in a matrix and the matrix of scores $\{f_{c,b}\}$ for ROIs is generated.

From this ROI score matrix, a vector of frame-level predictions is generated with

Multiple Instance Learning (MIL). It is assumed that at least one bounding box is corresponding to the object of interest. Log-Sum-Exp (LSE) aggregation (Ren et al., 2017) has been used to calculate the class score vector. This vector is transformed to probabilities with softmax operator.

González-Díaz et al. (2019) CNN can predict if the user is currently grasping the object or not. It is a multi-class classification problem with C+1 action classes, c=0 is the "No Grasp", and c=1...C for the actions of "Grasp of object c". A Long-Short Term Memory cells (LSTM) (Gers et al., 2000; Hochreiter et Schmidhuber, 1997) CNN have been used and the input of this network was created of consternation of four features: Magnitude of the gaze motion vector, magnitude of the ego-motion, distance of the gaze point to the center of the image, and vector of active object scores.

The input vector feeds a bottom LSTM layer with 256 units. The hidden state cells of this last layer are passed to a fully connected layer. The output is $s_n \in \mathbb{R}^{(C+1)x1}$ and this vector is converted to vector of probabilities with softmax layer. Also, a dropout layer with a factor of 0.5 has been used during the learning to reduce the overfitting.

This is an interesting exploratory research, nevertheless in our hybrid implementation we limit ourselves to the object to grasp detection and localization.

3.3.2 Dataset

The Grasping In The Wild (GITW) (LaBRI, 2016) dataset was recorded in the scenario of object-to-grasp recognition in prosthesis control by healthy volunteers. It is freely available for research at NAKALA CNRS server [1]. The GITW contains egocentric videos recorded by a camera on the eye tracker glasses. It includes the

[1] `https://www.labri.fr/projet/AIV/graspinginthewild.php`

gaze points of where the person was looking at each moment. The videos were recorded in the wild, in real kitchens, by different subjects. Every video corresponds to the situation in which the subject looks for an object and grasps it.

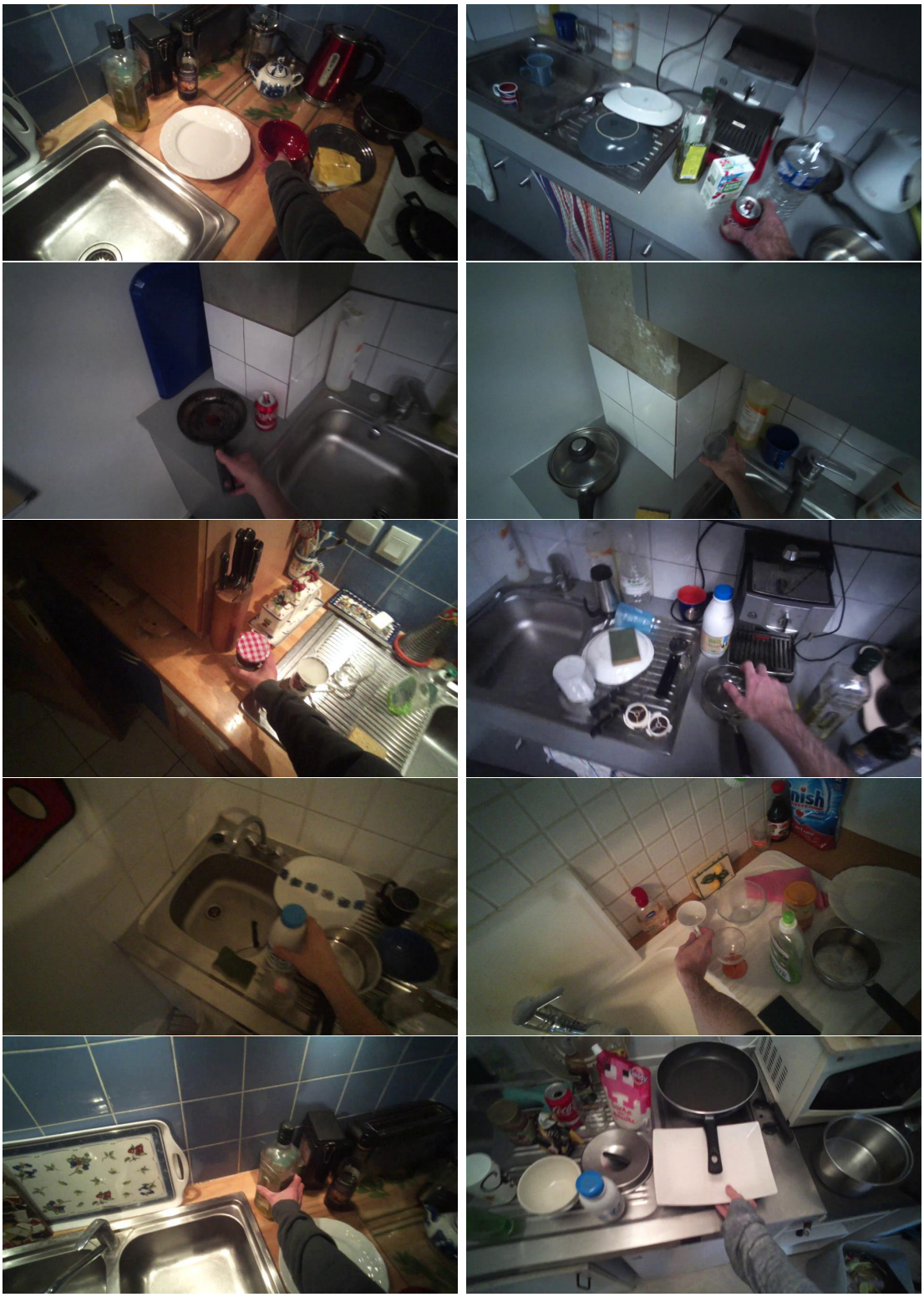


Figure 3.4: Examples of objects of the GITW(LaBRI, 2016) dataset.

Figure 3.4: Example object of the GITW LaBRI (2016) dataset.

The acquisition device used was Tobii Glasses 2 (eye tracker) with an egocentric scene camera. The Tobii Glasses video resolution is HD (1280 pixels × 720 pixels), and the video frame rate is 25 fps. The gaze point fixations are recorded at 50Hz frequency. The 16 different kitchen objects categories in the videos are: bowl, plate, wash liquid, vinegar bottle, milk bottle, oil bottle, glass, lid, saucepan, frying pan, and mug. Examples of these objects are presented in Figure 3.4. Different subjects recorded the dataset in five different kitchens. The videos were short, around 10 s long, as shown in Table 3.1 for the objects of bowl category.

Table 3.1: Bowl subdataset overview.

Number	Folder name	Duration (s)	Number of frames	bounding box objects
1	BowlPlace1Subject1	8.72	219	117
2	BowlPlace1Subject2	7.12	179	165
3	BowlPlace1Subject3	5.48	138	27
4	BowlPlace1Subject4	5.64	142	59
5	BowlPlace4Subject1	7.12	179	34
6	BowlPlace4Subject2	12.48	313	188
7	BowlPlace4Subject3	12.36	310	151
8	BowlPlace4Subject4	7.84	197	107
9	BowlPlace5Subject1	5.76	145	98
10	BowlPlace5Subject2	9	226	124
11	BowlPlace5Subject3	8.16	205	154
12	BowlPlace6Subject1	11	276	144
13	BowlPlace6Subject2	8.4	211	60
14	BowlPlace6Subject3	12.08	303	64
15	BowlPlace6Subject4	6.72	169	63
16	BowlPlace7Subject1	11.44	287	89
17	BowlPlace7Subject2	9.48	238	58
18	BowlPlace7Subject3	4.88	123	38
Total	N/A	153.68	3860	1740

The GITW (LaBRI, 2016) dataset contains 404 videos overall.

We carried out the time measurements on a subset of the GITW dataset, containing eighteen videos of “grasping a bowl" actions, recorded by four different subjects.

The kitchen environments are of different complexity, from a scene with just a few objects, such as the Can of Cola Place 2 Subject 3 videos (see Figure 3.6 a), to a highly cluttered scene, such as BowlPlace4, see Figure 3.6 b).

Figure 3.5: Example of the Bowl subset. It contains different bowls in different kitchens.

The "Bowl" subset illustrated in Figure 3.5 has been used in SIFT measurements. The reason was that this subset is very much representative of real-world scenes and diverse with light changes, cluttered and non-cluttered scenes, different object materials. Moreover, sometimes, we obtained strong blurring effects due to the camera motion, which was worn on the person's body.

Figure 3.5 shows that the materials and the colours of the objects are different. The first two kitchens have red and blue bowls, there are glass bowls in the third and the last kitchen. In the fourth kitchen there is a ceramic white bowl.

Figure 3.6: Example of non-cluttered and cluttered environment.

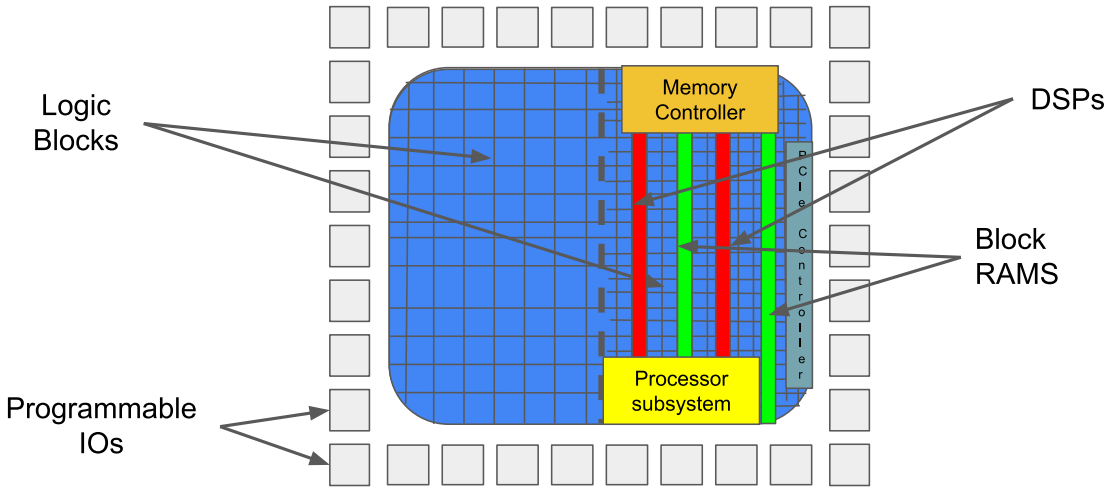


Figure 3.7: Basic FPGA architecture.

3.4 Target FPGA board

In the 1980s, the FPGA industry has been born and Xilinx introduced their first FPGA in 1984 (Trimberger, 2015), however the name FPGA was introduced later in 1988 after Actel popularized the term.

Application-Specific Integrated Circuit (ASIC) companies built-to-order custom integrated circuits in the 1980s. But those devices were just programmable one time. The big advantage of FPGA is that it can be reprogramable "infinite" number of times. That is a huge favor for FPGA because if something went wrong after the

ASIC chip was created, it has to start from the beginning. The FPGA just has to be reprogrammed, which is cheaper, and it is faster from a time point of view. So basically FPGA is a device where the engineer can create a new digital circuits just programming the device.

Xilinx XC2064 was the first FPGA, it contained only 64 logic blocks, each of which held three-input lookup table (LUT) and one register (flip-flop (FF)). Today, FPGAs, for example, Xilinx ZCU 102, which is described briefly in Chapter 3.4.1 contains 600K logic blocks. A general FPGA is shown in Figure 3.7. It contains logic blocks which can be further divided to LUTs and flip-flop (FF), programmable I/Os, dedicated memory modules (Block RAMs), digital signal processing (DSP) blocks, and direct multipliers. During the decades, the general architecture does not changed significantly. However, because of the improvement of the production technology, nowadays advanced DSPs and more logic blocks can be found in an FPGA chip.

A lot of different FPGA providers do exist in the world. Popular are AMD - Xilinx, Intel - Altera, Lattice, and Microsemi. The Xilinx has different types of FPGA families. The cheapest FPGA device families are the Spartans, the mid-class boards are the Zynq, and the high-end boards are the UltraScale, and many others.

In the next part of this Chapter 3.4.1 we will introduce the Xilinx ZCU102 and the Xilinx development software (in Chapter 3.4.2) and describe why we chose it for this research.

3.4.1 Xilinx Zynq UltraScale+ MPSoC ZCU102

During this research, the Xilinx Zynq UltraScale+ MPSoC ZCU102 (Xilinx, 2019) has been used. This device is suitable for accelerating computer vision and image processing algorithms.

The XCZU9EG FPGA device on the ZCU102 board has a processing system (PS) and a programmable logic (PL) part. The PS part has a quad-core Arm Cortex-A53, dual-core Cortex-R5F real-time processors, and a Mali-400 MP2 graphics processing unit. The Cortex-A53 is an Application Processing Unit (APU) to run OS and general purpose applications. The ZCU102 has a Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC chip. The Cortex-A53 is an Arm v8 architecture-based 64-bit quad-core multiprocessing CPU. The Cortex-R5 is a Real-time Processing Unit (RPU) and based on an Arm v7 architecture 32-bit RPU with a dedicated tightly coupled memory (TCM). The Mali-400 is a graphics processing unit with pixel and geometry processor and 64 KB L2 cache.

The device PS also has four high-speed serial I/O (HSSIO) interfaces. SATA 3.1 interface, source-only DisplayPort interface with video resolution up to 4K x 2K-30 (300 MHz pixel rate), USB 3.0 with 5 Gb/s line rate, Serial GMII interface-supports a 1 Gb/s SGMII interface, and Integrated block for PCI Express interface (PCIe) version 2.1.

Table 3.2: Xilinx Zynq UltraScale+ ZCU102 programmable logic resources.

Resource type	**Available**
BRAM	912
DSP	2,520
FF	548,160
LUT	274,080

The PL resources of the ZCU102 has been shown in Table 3.2. It has 912 block RAM (BRAM), 548,160 flip-flop (FF), 2,520 digital signal processing (DSP) units, and 274,080 lookup table (LUT).

The ZCU102 has other ports too. There are communication & networking ports such as RGMII communication 10,100 or 1000 MB/s, a serial GMII interface supports a 1 Gb/s SGMII interface, 4x SFP+ cage, SMA GTH, UART to USB bridge

RJ45 Ethernet connector Sata, PCIe Gen2x4 Root port. Those ports help the device communicate with other devices.

The ZCU102 has also some display port such as HDMI video input/output, External Retimer device driving an HDMI output connector, 9x GPIO user LEDs (8x PL 1x PS), VESA DisplayPort 1.2 source-only.

The PS and PL can be combined with multiple interfaces and DSP blocks to effectively integrate user-created hardware. They can also access memory resources in the processing system. The PS I/O peripherals, including the static/flash memory interfaces, share a multiplexed I/O (MIO) of up to 78 MIO pins. Zynq UltraScale+ MPSoCs can also use the I/O in the PL domain for many of the PS I/O peripherals. This is done through an Extended Multiplexed I/O interface (EMIO).and boots at power-up or reset.

The dissipation of the FPGA chip on the board is 20W maximum based on the thermal properties of the device package and heat sink installed on the chip (Xilinx, 2022a).

The ZCU102 can be used as an embedded device, because it can be booted from an SD card and can work in a standalone mode. A possible operating system is PetaLinux made by Xilinx. The PetaLinux has different packages such as Vitis and, Python. The Xilinx has created a software platform which is ideal for this project. In Vitis HLS the FPGA can be programmable in the high-level language C. More information about Vitis HLS can be found in Chapter 3.4.2.

3.4.2 Vitis HLS, IP and Kernel Flow

Vitis HLS (formerly Vivavo HLS) is a high-level synthesis tool that allows compilation of C, C++, and OpenCL functions to hardware modules using the device logic fabric and RAM/DSP blocks. Vitis HLS supports to develop a register-transfer level

(RTL) IP for Xilinx devices in C/C++ programming languages.

Vitis HLS automatically does the optimization in the C/C++ code. The code should not contain any recursion, because the compiler cannot synthesize that to a digital circuit. Thus, to achieve the best optimized RTL code, it is necessary to add some pragmas to the code. The pragmas help the compiler to optimize design, reduce latency, I/O ports usage, and resource usages. The pragmas are called directives in Vivado HLS.

The optimal process to design a logical circuit in the Vitis HLS (Xilinx, 2022b):

1. Compile, simulate, and debug the C/C++ algorithm.
2. View reports to analyze and optimize the design.
3. Synthesize the C algorithm into a RTL design.
4. Verify the RTL implementation using RTL co-simulation.
5. Package the RTL implementation into a compiled object file (.xo) extension, or export to an RTL IP.

It is possible to create different projects called "solutions" in Vitis HLS. The solutions are good for creating a digital circuit for different boards, or try different directives.

Advanced eXtensible Interface (AXI) 4 (Xilinx, 2017)is a part of ARM AMBA, a family of microcontroller buses. The first version of AXI was introduced in 1996. The current AXI4 was introduced in 2010.

It can handle 32-64 bit addresses, and the bus width is between 32-1024 bit. It has separate write address channel, read address channel, read data channel, write data channel and write response channel. The read and write channels are independent and transactions can be executed in parallel.

All AXI4 has a Master interface and a Slave interface. Xilinx implemented the AXI4 protocol as an IP.

The C/C++ codes are synthesized in the following way to a RTL description:

- The Top-level function and the sub-functions from this top-level function are designed the digital circuit. The top-level function arguments can be mapped with the directive (#pragma INTERFACE) to the I/O ports and communicate the PL and PS part using the AXI bus (for more information, please see Chapter 3.4.1).

- The sub-functions of the top-level function are synthesized into blocks in the hierarchy of the RTL design. The RTL design and the original top-level function hierarchy of modules will be identical after the synthesis. The #pragma INLINE directive can disable the automatic inlineing of the selected sub-function. If the Vitis HLS compiler thinks inlineing a sub-function will cause better performance, then the automatically generated version The #pragma ALLOCATION directive

- Loops in the top-level and sub-functions are kept rolled and are pipelined by default to improve the performance. The #pragma UNROLL directive can unroll the loops manually, the compiler will unroll the loops if it improves the performance of the RTL design, for example unrolling nested loops. Loops can be pipelined with a finite-state machine fine-grain implementation (loop pipelining with #pragma PIPELINE directive) or with a more coarse-grain handshake-based implementation (dataflow with the #pragma DATAFLOW directive).

- Arrays in the C/C++ code are synthesized into memory such as block RAM (BRAM), lookup table (LUT) RAM, or UltraRAM in the RTL design. With

directives for example #pragma ARRAY_PARTITION or #pragma ARRAY_RESHAPE it, the type of memory or the read/write memory transfers per clock cycle can be changed. If an array is a part of the top-level function interface, then the array is implemented as ports with access to a block RAM outside the design.

After synthesis has been done, a report is generated in the Vitis HLS. This report contains the generated RTL design modules, resource usages, and the computational cost. The different solution results can also be compared to each other in the Vitis HLS.

If the synthesis was successful, then an IP block can be generated in the IP flow. After the IP is generated, an embedded system can be created in the Vivado IP integrator. That means that the generated digital circuit is connected to the other parts of the system via AXI4 buses such as memories, CPUs, other IPs, and peripheries. The bit stream file is generated based on the created embedded system in the Vivado IP integrator.

The generated bitstream alone is not enough for running on the Xilinx device. An application is required to control the operation of the system, which is running on the Xilinx SoC embedded Arm CPU. For this application, there are two solutions: Vitis SDK and PYNQ. Vitis SDK supports C/C++, while PYNQ supports Python. The operating system of the embedded system can be selected in Vitis SDK. The operating system can be standalone, Linux or FreeRTOS. The other method (PYNQ) will be introduced briefly in the Chapter 3.4.3.

3.4.3 PYNQ

PYNQ (Xilinx, 2018a) is a Xilinx open-source project. PYNQ (Xilinx, 2018a) offers a Jupyter notebook based framework with Python APIs for using Xilinx platforms. It is supported by several platforms such as Zynq and Zynq UltraScale+, Zynq RF-

SoC, Alveo and AWS-F1 instances. It allows for designing embedded systems to use Zynq devices, without having to use ASIC-style design tools to design programmable logic circuits.

The FPGA part of the Zynq devices is called programmable logic (PL). In PYNQ, the logic circuits in the PL are presented as hardware libraries called overlays. These overlays act in the same manner as regular software libraries. The overlays can be used as a black box and accessed through a Python API. The overlays are reusable and reconfigurable, just like regular software libraries. However to create a new overlay the knowledge of designing porgrammable logic circuits is still required.

The ZYNQ SoC embedded processors and the PL are programmed in with Python in PYNQ. PYNQ uses CPython which is written in C, and integrates a large amount of C libraries. It can be extended with optimized code written in C as well. It is possible to use C programming language if it is more suitable for the task in PYNQ.

PYNQ is a web-based architecture that is also browser-independent. The built-in Jupyter notebook runs an Interactive Python (IPython) kernel and a web server directly on the ARM processor of the Zynq devices. The web server brokers access to the kernel via a suite of browser-based tools that provide a dashboard, bash terminal, code editors, and Jupyter notebooks.

We choose PYNQ because during our embedded system design, it is possible to use a high-level language such as Python. The implemented overlay is used as a hardware library. The web-based architecture from the embedded processors makes it possible to create an embedded system easily.

3.4.4 Vitis AI

Vitis AI (Kathail, 2020; Xilinx, 2022c) can accelerate AI inference on Xilinx hardware platforms such as FPGAs, SoCs, and Versal Adaptive Compute Acceleration Platforms (ACAP). The development environment includes of optimized IP cores, tools, libraries, models, and example designs. It makes possible to accelerate a Neural Network on FPGA without special FPGA knowledge.

Vitis AI supports different deep learning frameworks such as PyTorch and TensorFlow.

The Vitis Model Zoo contains the optimized deep learning models, which are usable on Xilinx platforms. The models include different fields of computer science such video surveillance, robotics and data center. From an accelerated network point of view, there are 4 different types:

- Classification: VGGnet, ResNet, Inception, Mobilenet
- Detection: Light head R-CNN, SSD, YOLO V2, YOLO V3
- Segmentation: Enet, Segnet, ESpnet, FPN, Deeplab V3+
- Pose estimation: Openpose, Coordinates regression

The Vitis Model Zoo also contains some information about the model:

- backbone, the name of the network, for example ResNet
- Input size, the size of the required data
- FLOPs number of floating-point operation in a second
- Parameters, i.e. the parameters of the network

and training set, validation set, platform, the accuracy of the floating- and fixed-point models.

Deep Learning Processor Unit (DPU) is a programmable engine optimized for deep neural networks. It is built up from different parameterizable IP cores. For those IP cores, no place and route are required. It can accelerate different kind of algorithms such as computer vision for example classification or detection. The DPU has a Vitis AI specialized instruction set. This is facilitating the efficient implementation of deep learning networks.

The DPU allows accelerating several networks such as VGGnet, ResNet and SSD, among others. The DPU is scalable to fit Zynq UltraScale+ MPSoCs (like ZCU 102) and other Xilinx products.

Quantization and channel pruning techniques are suitable in reducing the amount of computation and required memory bandwidth. This is ideal for an embedded system that has limited resources. It also helps to achieve low-latency and high-throughput. Quantization and channel pruning cause very little effect on accuracy.

Neural networks use 32-bit floating-point weights and activation values during training. The Vitis AI converts those 32-bit floating-point to 8-bit integer (INT8) to reduce the computational complexity without losing prediction accuracy. The Vitis AI quantizer supports quantizing the convolution, pooling, fully connected layer operations, and batch normalization Xilinx (2022c). The generated fixed-point network compared to the floating-point network has better computational performance, higher power efficiency, and it needs less memory bandwidth.

Post-training quantization (PTQ) is possible, and it requires a small set of unlabelled images to analyze the distribution of activations. PTQ causes little accuracy drop after quantization. Nevertheless, PTQ causes a higher accuracy loss in MobileNet as reported in Xilinx (2022c).

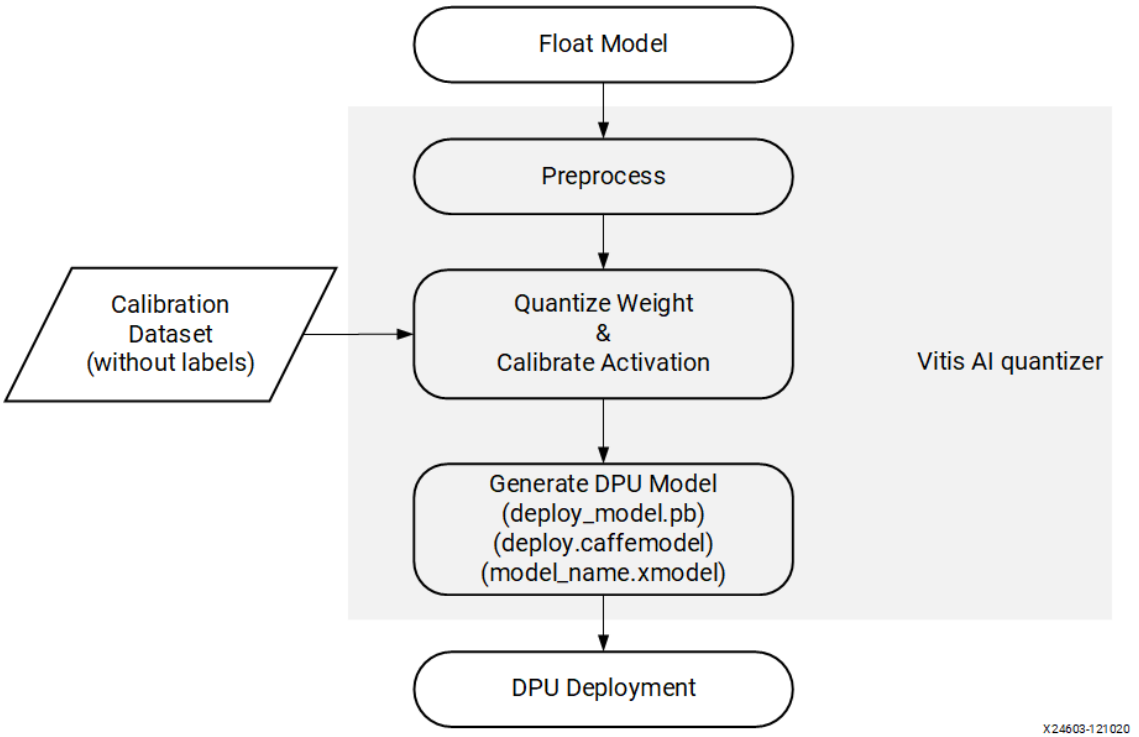


Figure 3.8: Vitis AI quantization process Xilinx (2022c).

Figure 3.8 shows the chain of the quantization in Vitis AI. The input of the Vitis AI quantizer is a floating point network. After the preprocessing the Vitis AI quantizes the weights and biases and activations to the given bit width.

There is an optional step before quantization, the inspector. The information partition tells which operators will run on DPU and which one should run on CPU.

To improve the accuracy of the network Vitis AI quantizer must run several iterations of inference to calibrate the activations. After that calibration, the quantized model is transformed into a DPU deployable model.

3.5 Critical analysis of computation complexity

In the first half of this section, the computational time measurements of the existing software solution will be shown.The results were analysed to show whether the

module was suitable for real-time processing or not.

The second half of the section contains our proposals to accelerate the software solution.

3.5.1 Time measurement of the whole system

Our goal is to build up a real-time neuroprosthetic arm control. In our case, that means the whole system should process a frame in less than 100ms. There is another constraint, namely that each module must process data in 40ms (25fps). The Tobii eyetracker can record 25 frames per a second, which means that we can use pipeling technique to accelerate the system.

Table 3.3 illustrates the average computational time of the system in milliseconds. The first column contains the module name, and the second column contains the Intel i5 7300HQ (Intel, 2017) CPU results. In the third column, the ARM A53 (Xilinx, 2021a)-embedded CPU results are given.

The Intel i5 7300HQ CPU can compute the following modules less than 40ms: FLANN matcher, homography estimation, Gaze point projection, bounding box generation, Faster R-CNN and the MIL Aggregation. The SIFT keypoint extraction takes 72.407 $\pm$ 3.349 ms, which is too much computation time for a module. The Intel i5 can compute the KDE estimation on a frame in 12.477 $\pm$ 23.306ms. The variance of the KDE estimation is too much (23.306 ms), it is more than the average computation time (12.477 ms). The computation time of the ResNet50 is higher (89.952 $\pm$ 2.568ms), than the required 40ms. The total computation time is 202.521 $\pm$ 29.966 ms in the Intel i5 7300HQ, which is 4.93 fps. This is higher than the maximum allowed 100ms computational time of the system.

The Intel i5 7300HQ power consumption is 45W. This is too high for a wearable device. Because of that, the system also tested it on an ARM A53 embedded CPU,

Table 3.3: The average computational time measurement of the whole system on a regular computer and an embedded system. The goal is a system which can process a frame less than 100ms and each module maximum computation time should be less than 40ms. The SIFT, KDE estimation, ResNet50, and the Faster R-CNN computation times are too high for real-time processing.

	Computational time / frame (ms)	
Module name	**Intel i5 7300HQ CPU**	**ARM A53**
SIFT Lowe (2004)	**72.407 ± 3.349**	**865.499 ± 8.437**
FLANN matcher	3.094 ± 0.638	18.223 ± 3.867
homography estimation	0.270 ± 0.075	2.359 ± 0.778
Gaze point projection	0.015 ± 10^{-4}	0.089 ± 0.003
KDE estimation	**12.477 ± 23.306**	**126.672 ± 238.900**
bounding box generation	0.424 ± 0.020	2.659 ± 0.027
ResNet50 (He et al., 2016)	**89.952 ± 2.568**	**1800.327 ± 17.915**
Faster R-CNN Ren et al. (2017)	23.718 ± 0.010	**285.121 ± 0.002**
MIL Aggregation	0.164 ± 10^{-6}	0.727 ± 10^{-6}
Total time (ms)	**202.521 ± 29.966**	**3099.017 ± 269.927**

which has a low 5.6W power consumption.

The bottleneck modules of the whole chain of the system are the SIFT, KDE estimation, ResNet50 and the Faster R-CNN computation on the ARM53. To extract the SIFT keypoint takes 865.499 ± 8.438ms on the ARM A53 CPU. The KDE estimation takes 126.672 ± 238.900ms, the high variance is also a problem, similarly to the other experiment on the Intel i5 7300HQ CPU. The ResNet50 feature extraction takes 1800.327 ± 17.915ms on an ARM A53 CPU, which is too slow for our target computational time. The Faster R-CNN module computation time for a frame is 285.121 ± 0.002 ms and this should be accelerated for real-time processing. The processing speed of the ARM A53 (Xilinx, 2021a) embedded CPU is slower (3099.017 ± 269.927 ms) than the Intel i5 7300HQ (202.521 ± 29.966 ms), however it is more suitable for a wearable device platform.

The measurements show that the current setup with the whole chain of modules is not yet suitable for real-time processing. However, there are methods to accelerate the slow modules with optimization of the algorithms. Thus, accelerating some modules on FPGA and with pipelining the modules, with some delays, the real-time processing speed is achievable. A hybrid HW/SW solution of this system is introduced in the next subsection, based on the computational time measurement.

3.5.2 System hybridization

To propose a hybridization of the system, compatible with real-time performance, we have conducted thorough time measurements on different CPUs to identify the most time-critical modules, see in Table 3.3.

Kóta et al. (2019); Han et Oruklu (2014) showed that a hybrid embedded system is good to solve computer vision and image processing algorithms and to accelerate the algorithm for real-time processing speed. They also showed that these embedded

systems are energy efficient and wearable, which is necessary in our case to control a robotic arm.

Our measurements show, that one the bottleneck is the Scale-Invariant Feature Transform (SIFT) detector, which is required in our system for geometric alignment of gaze points—see Figure 3.2. The main steps of the SIFT are the following: scale-space extrema detection, keypoint localization, orientation assignment, and descriptor generation. For hardware acceleration, we have chosen Xilinx UltraScale ZCU102 (Xilinx, 2019) FPGA as it supports the parallel execution, and the energy consumption is very low.

The gaze point noise reduction module is based on the KDE estimation. The variance of the computational time is too high for the KDE estimation, as Table 3.3 shown. To reduce the required computation time, the outlier gaze point should be eliminated with a clustering algorithm like the Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

The other complex module is the CNN for object recognition. Nevertheless, CNN is pre-trained offline for a given set of object categories. The spatial regularity of the CNN inference makes it ideal for FPGA implementation, and hundreds of papers have been published in this area in recent years. The proposed solutions can be divided into two classes: streaming architectures and parametrizable blocks.

The structure of the streaming architectures closely follows the data flow of the given network by connecting templated processing blocks in a pipeline. Input and output of the blocks are data streams (FIFO interfaces) and each operation in the network—e.g., convolution, pooling, nonlinear response, etc.—has a dedicated block for FPGA implementation (Blott et al., 2018).

The usual template parameters in the case of a convolution block are the number of input and output layers and the size of the convolution window. The input image

is fed into the system in a row-wise order, which makes it possible to connect the network directly to a camera input. The latency of the resulting system is low because the convolution blocks can start processing as soon as the first rows required for the computation are available.

The main drawback of the streaming architecture is that all the weights for the computation must be stored on-chip, which is not possible for large networks. In addition, the computation load of the layers is very different. Therefore, different design optimization strategies must be used for each layer, which makes the design process complicated.

Another approach is to use a compiler to break down the entire CNN computation into a series of tensor operations and create parametrizable hardware blocks to efficiently execute them (Xilinx, 2021b; Moreau et al., 2019). The fundamental building block of these architectures is a matrix–matrix multiplication block, which is usually extended by an additional functional unit to efficiently carry out other operations, such as max pooling and nonlinear transformation. The matrix–matrix multiplication is usually carried out by a systolic array of Multiply–Accumulate (MAC) units. A critical part of the system is the compiler, which is also responsible for the optimal scheduling of the tensor operations. The input image, network weights, and partial results are stored in off-chip memory, so the network size is not limited by the size of the FPGA device. On the other hand, the latency of the CNN computation is higher in this case because the entire image frame must be captured and stored in the memory before processing is started. Performance of the system might be also limited by the available off-chip memory bandwidth.

Taking into account the real-time constraints and also the power dissipation, we implement a hybrid solution both for the preliminary processing steps before feeding gaze-driven CNN and the CNN as well. Referring to Figure 3.2, the hybridization

of the preliminary steps is given in Table 3.4.

Table 3.4: Hybridization of preliminary steps in the pipeline, which contains two main blocks: Gaze Point Alignment Block and Gaze Point Noise Reduction Block and its sub-modules.

Module	CPU	FPGA
Gaze-Point Alignment Block		
SIFT Detection Fejér et al. (2021a)	-	X
SIFT Matching	X	-
homography estimation	X	-
Gaze-point projection	X	-
Gaze-Point Noise Reduction Block		
KDE estimation	X	-

As for the gaze-driven CNN implementation, accordingly with the time measures for real-time compatibility and simplification of R-CNN input by channel number reduction we proposed—see Section 3.3.1—only the ResNet backbone is implemented on FPGA; as depicted in Figure 6.1. The details of all modules from the input of CNN to the final aggregation of decisions by MIL are given in Table 3.5 below.

Table 3.5: Hybridization of the gaze-driven CNN.

Module	CPU	FPGA
ResNet50	-	X
Reduction Layer	X	-
Faster R-CNN	X	-
MIL aggregation	X	-

The reference software implementation of the system was executed on a four-core Intel i5 7300HQ (Intel, 2017) laptop CPU running at 2.5 GHz. This software system is also compiled for the four-core ARM Cortex A53 (Xilinx, 2021a) processing system (PS) of the Xilinx Zynq UltraScale+ XCZU9EG device on the ZCU102 development

board. Based on these measurements, we propose a system was partitioned between the PS and the PL parts of the device. Specialized accelerator circuits were designed for the modules of the proposed system, which cannot be executed fast enough on ARM Cortex A53 processors. A traditional register-transfer-level (RTL)-based design of a digital circuit is time-consuming; therefore, the Xilinx Vitis HLS system was used to create the FPGA-based circuits from a high-level C/C++ description.

3.6 Conclusion

Hence, in this Chapter 3 we have presented the whole approach for recognition of the object-to-grasp in the ego-centric camera view for prosthesis servoing scenario.

We have introduced the GITW video dataset which was recorded for this purpose and is used in our work.

Xilinx Zynq UltraScale+ MPSoC ZCU102 FPGA board has been described, and we choose this research, because the FPGA is a good choice for accelerating computer vision algorithm. It has low power consumption too, which is good for creating a wearable device. Xilinx created a development environment (Vitis HLS, Vitis AI, PYNQ) to accelerate the time-critical algorithms.

We have introduced the Gaze Driven Faster R-CNN that can predict the object location, type. It can also predict if there is a grasping action or not.

We have analyzed time complexity of each block of the whole object recognition chain.

And finally, we have proposed the hybridization scheme.

In the following chapters we will describe implementation, either on FPGA or on embedded CPU, of each block of the chain.

In the next Chapter 4, we will focus on the implementation of SIFT point de-

tector.

Chapter 4

Hybrid solutions for SIFT detector implementation

4.1 Introduction

In computer vision and image processing, it is important to extract the characteristic keypoints in an image in different image analysis tasks. It is useful for object matching for finding an object in an image like a banknote recognizer (Solymár et al., 2011), or scene matching in stereo vision as for example in a real-time multi-camera vision system for UAV collision warning and navigation (Zarándy et al., 2016). In our case, we want to match the consecutive frames to project the gaze points to the actual frame and estimate a smooth position of a gaze point on the object.

Several keypoint detectors of keypoints do exit today such as SIFT (Lowe, 2004), SURF (Bay et al., 2008), ORB (Rublee et al., 2011) (please find a detailed comparison of those algorithms in Chapter 2.4.2). We have chosen the SIFT algorithm to implement on FPGA because Karami et al. (2017) have shown that in different kinds of transformations and deformations such as scaling, rotation, noise, fish-eye distortion, and shearing SIFT outperformed other methods in terms of precision.

However, our experiments for computational time measurement, Table 3.3, show

that the SIFT CPU implementation is not fast enough for real-time processing of egocentric video. It is necessary to accelerate this algorithm for controlling a robotic arm in real-time. Without that, the controlling algorithm will be too slow and cause discomfort for the user.

Different hardware accelerators do exist in the market like GPU and FPGA. We compared these accelerators in Chapter 2.5 and we have decided to accelerate SIFT algorithm on FPGA. It is a good solution for creating a wearable device with high computational speed.

Our first scientific contribution consists in accelerating the SIFT algorithm on FPGA. Obviously, FPGA SIFT has to be fast enough for real-time processing, but it is also important to get low energy usage. As far as the accuracy of the solution is concerned, it has to give the same result as the CPU version.

4.2 Base-Line SIFT algorithm

In the first half of this section, we will discuss and describe the steps of the original SIFT (Lowe, 2004) algorithm. The second half of this section is about the solution and limitations of the FPGA SIFT (Lowe, 2004) implementation.

4.2.1 SIFT algorithm

Scale-Invariant Feature Transform method was proposed by Lowe (2004) and originally developed for image stitching to solve the problem in rotation-, scaling-, affine deformation-, viewpoint change-invariant way, remaining robust to noise, and illumination changes when different images of the same scene are matched. It is therefore useful for matching of video frames in order to estimate transformation parameters between current frame and previous frames in our problem.

The SIFT algorithm has two main steps: i) the keypoint detection and the ii) descriptor computation. The keypoints are extracted from images to match first. Then invariant descriptors are computed for detected points and the matching process consists in comparison of descriptors. Keypoint detection and descriptor computation comprise several sub-steps, which are depicted in Figure 4.1. These steps are:

1. scale-space extrema detection
2. keypoint localization
3. orientation assignment
4. descriptor computation

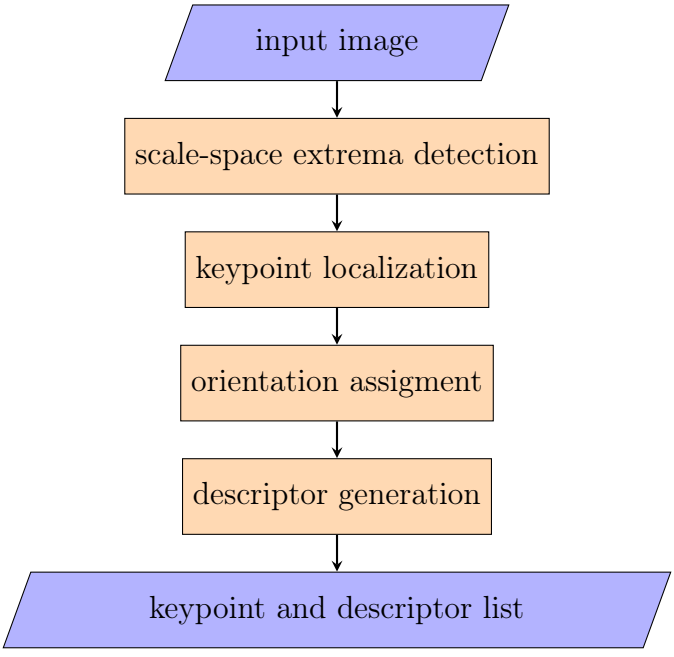


Figure 4.1: The main steps of the SIFT(Lowe, 2004) algorithm.

Next subsections contain the detailed description of the SIFT algorithm.

Scale-space extrema (SSE) detection

The Input Image denoted by $I(x, y)$ is convolved by a Gaussian kernel $G(x, y, \sigma)$ resulting in the image of the scale-space $L(x, y, \sigma)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{4.1}$$

where * is the convolution operator in x and y, and $G(x, y, \sigma)$ is defined by the following equation:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{4.2}$$

Here σ is the scale parameter. The next step is to detect candidate keypoint locations. The scale-space extrema in the Difference of Gaussians (DoG) functions denoted by $D(x, y, \sigma)$. The DoG is computed as the difference of two consecutive scales, which are separated by a constant factor k in a scale-space, see equation 4.3.

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{4.3}$$

The scale-space is split into octaves. After all the scale-space images have been computed in one octave, the scale-space image of the first scale of the next octave is calculated as a sub-sampled version of the last scale-space image in the previous octave.

For computing the local maximums and local minimums in a scale-space, three consecutive DoG images in an octave are needed. As illustrated in Figure 4.4 a pixel in a DoG image, marked by X is compared to its $3 \times 3 \times 3$ neighbourhood in the current and two neighbouring scales. If the pixel is a maximum or a minimum, then

the pixel is considered as a candidate keypoint.

The original SIFT implementation (Lowe, 2004) used the following hyperparameters in the scale-space extrema detection part: initial σ was set to 1.6, the number of octaves was 4, the number of intervals (scales) in each octave was 5, and the parameter k was set to $\sqrt{2}$.

Keypoint localization

In this stage, the candidate keypoints localization is determined and a set of keypoints is filtered by eliminating the parasitic keypoints with low contrast or points on edges.

In Lowe (2004) Taylor series expansion has been used to get a more accurate localization of the candidate keypoint. The expansion of the function D(x,y,σ) is computed around the (x, y) candidate keypoint. The precise position of the extremum (DoG position) is then calculated from the derivative of this Taylor expansion.

To filter our parasite keypoints candidates keypoint DoG response is compared to a threshold. If the magnitude of the maximum or minimum of DoG function in the keypoint is smaller than the given threshold (e.g. 0.03 as in reference implementation (Lowe, 2004)), then the candidate keypoint is rejected.

The DoG response is also strong on the image edges. Candidate keypoints situated on an edge should be removed, as the edges often have aliasing effects and these keypoints are not stable. To discard those candidates keypoints Hessian matrix H is used to determine the principal curvature around the given DoG point.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{4.4}$$

Here D are 2^{nd} order partial derivatives. The curvature is computed considering

the eigenvalues of the Hessian matrix. Let α and β be the two eigenvalues of the matrix. In that case, the sum and the product of the eigenvalues are:

$$\begin{aligned} Tr(H) &= D_{xx} + D_{yy} = \alpha + \beta \\ Det(H) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \end{aligned} \tag{4.5}$$

If the determinant of H is less than 0, then the curvatures have different signs, so the point is a parasite keypoint and has to be discarded.

Supposing that α is the largest magnitude eigenvalue and denoting by r the ratio between the largest magnitude eigenvalue and the lower one, then $\alpha = r\beta$, in that case:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{4.6}$$

Now we can check the ratio of principal curvatures if it is below some threshold or not:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{4.7}$$

If the ratio is larger than the given threshold, the point is a parasitic keypoint on the border and is discarded. In original implementation (Lowe, 2004) r=10 was used.

The last two steps concern SIFT descriptor computation. The first step consists in assignment of the orientation to the keypoint to further use it in the rotation-invariant descriptor computation. The last one is the descriptor computation per see, which represents a histogram of orientations of gradients in the vicinity of a detected point, weighted by the magnitude of the gradient. In our solution, these two steps are implemented with a reference software on a host computer accordingly to the original algorithm, and we do not detail them here.

4.2.2 Constraints for FPGA implementation

A FPGA board has limited available resources. Because of that, it is possible to implement just the time-consuming parts of the algorithm to an FPGA, and the other parts of the algorithm can run on the embedded CPU as an embedded HW/SW system. For example Chang et al. (2013) proposed an embedded HW/SW system, where the Gaussian Pyramid is computing on the FPGA part, and the rest of the algorithm in the embedded CPU.

It is another solution to reduce the FPGA resource usage is to simplify the SIFT algorithm. Shao et al. (2015) changed the Gaussian pyramid building, instead of filtering in parallel with different size filters they cascaded several smaller sizes Gaussian kernels. The design of the system is simplified but requires more memory resources.

The required resource demand on FPGA can be reduced to change the number representation from floating-point to fix-point (Vourvoulakis et al., 2017, 2016).

For more details of the different FPGA SIFT implementation please find in Chapter 2.4.2.

We have decided to simplify the SIFT algorithm, for reducing the resource demand of our implementation. Instead of the Taylor expansion for better key-point localization, we remain with the original precision of localization, but remove parasite key-points. Hence, we implemented a Non-maximum Supression algorithm. Indeed, the Taylor expansion hardware solution would require divider circuits which have a high FPGA resource demand.

4.3 Hybrid SIFT implementation

In this section, we present our hybrid hardware/software implementation of SIFT detector and descriptor computation and focus on the parts which have been implemented in FPGA.

First of all execution time of the SIFT algorithm using the GITW dataset (LaBRI, 2016) was measured on the ARM Cortex-A53 CPU of the Xilinx ZCU102 board and on an Intel Xeon E5-2620 server CPU for reference. Average execution time for the SIFT keypoints detection part was 193ms and 46ms on the ARM Cortex-A53 CPU and the Intel Xeon E5-2620 CPU respectively. The descriptor computation part can be performed faster. It is executed in 62ms and 17ms on the CPU architectures. Based on the time required to execute different steps of the algorithm the system was partitioned between the FPGA and the embedded microprocessor. Considering these experimental results the bottleneck of the algorithm is the keypoints detector part, therefore it is selected for acceleration on the FPGA as it is illustrated in Figure 4.2.

Figure 4.2 shows our solution. The first four steps: Gaussian Filtering and Difference of Gaussians computation (GFDG), Scale-Space Extrema Search (SSE), Non-maximum Supression (NMS), filtering out points on edges, which we call "Edge Detection" for simplification (ED). These four steps constitute the SIFT point detection part. It is running on the FPGA. The SIFT descriptor generation part is running on the CPU. The CPU and the FPGA SIFT module are communicating using the Advanced Microcontroller Bus Architecture Advanced eXtensible Interface 4 (AMBA AXI4) (Xilinx, 2017) protocol.

The development of our FPGA based SIFT implementation is founded on two open source solutions: the OpenCV SIFT function (Bradski, 2000) and the Open-

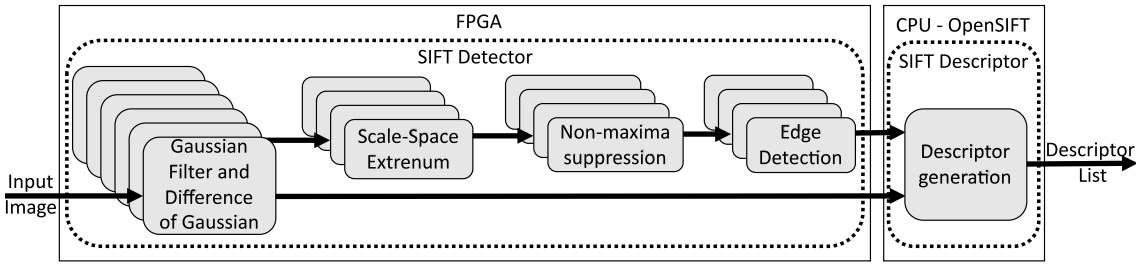


Figure 4.2: The block diagram of our proposed architecture. The SIFT detector part contains the GFDG computation, SSE, NMS and Edge Detection modules. The SIFT descriptor part contains the orientation assignment and other descriptor computation steps.

SIFT library (Hess, 2010). They were used as a reference code.

4.3.1 GFDG computation module

This module computes the Gaussian filtered images (see eq. 4.1) and the DoG (see eq. 4.3) in an octave. Also handling of the border extension (padding) is performed here. It comprises delay arrays which are required for the synchronization of partial results.

The initial scale parameter σ in a Gaussian filter (see eq. 4.2) is a predefined value and from this the weights of the Gaussian kernel are computed. The Gaussian kernel size is the size of the convolution kernel, of 5×5 in our case. During the Gaussian computation the method proposed by Shao et al. (2015)is used. Several Gaussian kernel computations are cascaded in a pipeline and each stage is working on the result of the previous stage. Thus, the main input of each pipeline module is the image filtered in the previous stage (or the original image in case of the first stage). For synchronization purposes the results of the previous stages (Gaussian filtered images, DoG images) are also loaded and buffered. The outputs of the module are: the filtered image, the DoG between the image filtered in the previous and the current stage. The buffered results from the previous stages are also sent to the next stage. After the GFDG module all these results will be processed by the

SSE module. The results computed by the different stages of the GFDG module can be synchronized utilising this structure.

The border adder/padding adds a frame around the input image. This is necessary for convolution computation at border pixels. One of the OpenCV (Bradski, 2000) border padding is called BORDER_REFLECT_101 and performs the padding by a mirror reflection which is used in our reference OpenSIFT (Hess, 2010) implementation.

In that case, if the input image is abcdefgh then the output is: gfedcb|0abcdefgh|0gfedcba. Therefore, the implementation uses 5×5 kernel in the Gaussian convolution, two extra rows and columns are added around the image. In our FPGA implementation we also perform the same mirror padding.

The GFDG computation module can convolve an input image with a given Gaussian kernel and subtract the result from the input image to compute the Difference of Gaussians (DoG). To utilize the computing power of the FPGA the 2D sliding windows technique is used similarly as in Vörösházi et al. (2008). In a general case when the size of the window is $N \times N$ and the size of the image is $W \times H$, the last $N - 1$ rows from the image must be stored on the FPGA in an $(N - 1) \times W$ sized array. The process is illustrated in Figure 4.3 where the kernel size is $N = 3$, and the image width $W = 5$ and height $H = 6$ for the sake of simplicity.

In a single clock cycle a new pixel data can be written into and an old pixel data can be read from the row memory in parallel with the computation of a new result. This memory arrangement makes feasible pipeline processing of the image.

The input image data of the Gaussian filter is stored in input image window which reads data from the temporary memory as shown in Figure 4.3. The resulting array can be found below the Gaussian filter. The green squares indicate the data which will be read next time, the red square indicates the being currently processed

and written data and the blue squares indicate the currently read data.

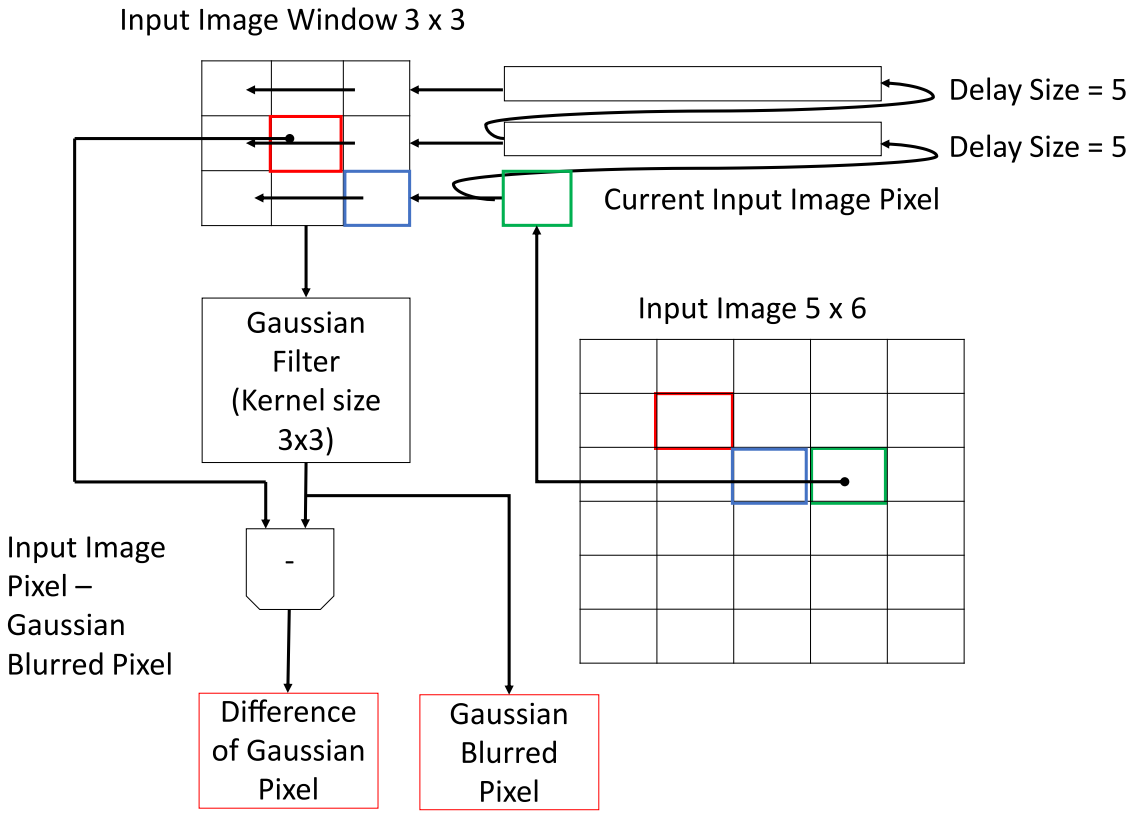


Figure 4.3: Gaussian Filtering and Difference of Gaussians computation (GFDG). The red rectangle is the current computed pixel. The green rectangle is the current input pixel from the input stream. Neighbours of the currently computed pixel are stored in the input image window. Two lines of the most recently used pixels are stored in the "temporary row delay arrays". Gaussian blurred image and the Difference of Gaussian image are the outputs of the module.

4.3.2 Scale-space extrema search module

The scale-space extrema detection module works in the following way. The current pixel absolute value in $3 \times 3 \times 3$ volume (showed in Figure 4.4) is checked if it is higher than a predefined contrast threshold (see section 4.4). If the pixel value is greater than the threshold then the next step is to check that the given point is higher or smaller than 0. If it is larger than 0 and the pixel is a maximum in the

given range of $3 \times 3 \times 3$, then the point is a candidate keypoint. If the given pixel is less than 0 than the current pixel is checked whether it is a minimum or not. If it is the minimum then the point is a candidate keypoint. If the given point is neither the maximum nor the minimum, the candidate point is rejected.

Similar 2D windowing technique is used in this module as in the GFDG module (see Figure 4.3) to efficiently generate the neighbourhood of a pixel. Synchronized results of the preceding three DoG computations are stored in a $2 \times 3 \times W$ row buffer to generate the $3 \times 3 \times 3$ window around each processed pixel. The Gaussian filter computation is replaced by the threshold detection circuit to mark candidate keypoints.

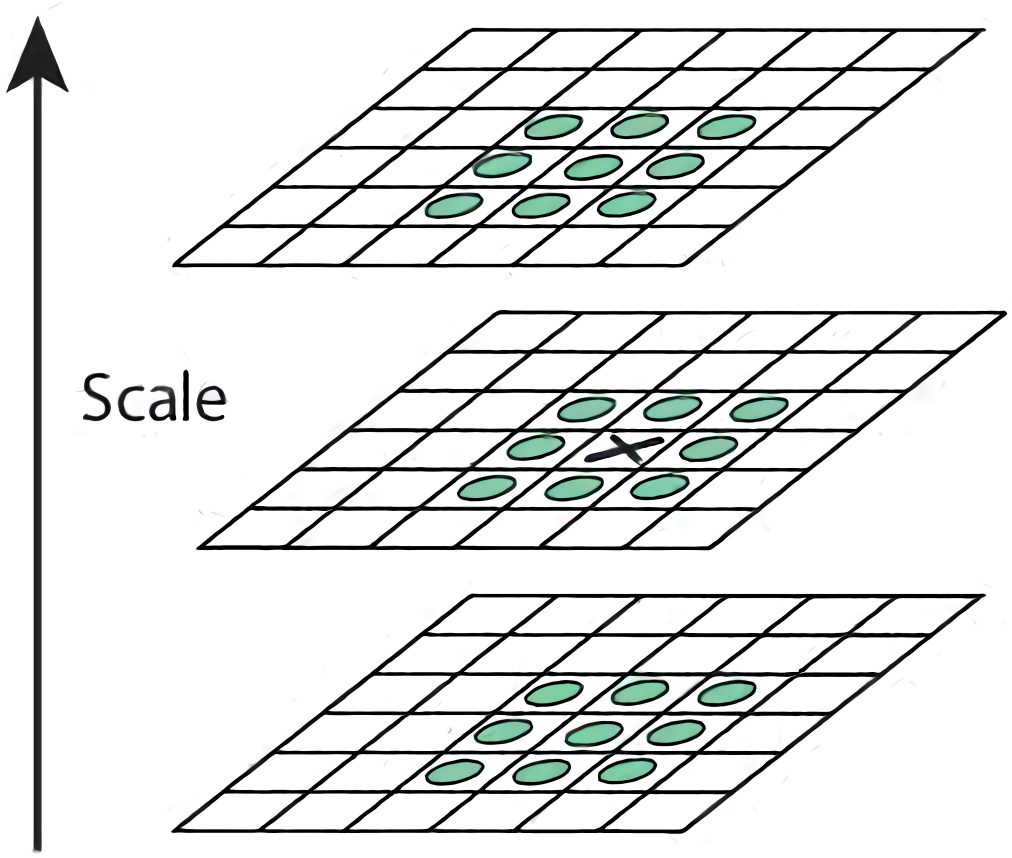


Figure 4.4: Calculate the local extremas in one scale (Lowe, 2004).

4.3.3 Non-Maximum Suppression module

In this subsection the simplification of the SIFT point detection algorithm is presented that is proposed for the FPGA. It consists of the choice of location of a relevant keypoint by a Non-maximum Supression (NMS). Instead of computing Taylor expansion (in keypoint localization step) in our FPGA implementation a NMS is proposed. The principle is based on the experimental analysis of detected keypoints with standard settings. It has been shown that many parasitic points have been detected despite filtering in the vicinity. This filtering was done at important (in the sense of magnitude value) SIFT points situated on the details of images. Furthermore, for matching of video frames, we do not need sub-pixel accuracy which is proposed by Lowe (2004) when computing Taylor expansion of the DoG and searching for its extrema. NMS can be computed by using absolute value and comparison operations only and no multipliers are required as in the case of Taylor series expansion. This simplification reduces the number of FPGA resources required to implement this part of the algorithm. We can simply express this algorithm as follows.

Lets us consider a given detected point x_{dp} on a discrete scale-image grid and its DoG Response value $D(x_{dp})$. Let us consider $\Omega(x_{dp})$ a $n \times n$ neighbourhood of x_{dp} in the current octave. We call it NMS kernel. Let us denote with th_{DoG} a threshold value which suppress low activation value. Let us introduce a function $f : \Omega \longrightarrow \mathbb{R}$:

$$f(x_{dp}) = \begin{cases} 1 \quad \text{if} \quad \forall x \in \Omega \Rightarrow |D(x_{dp})| \geq |D(x)| \\ \qquad\qquad \cap |D(x_{dp})| > th_{DoG} \\ 0 \qquad\qquad \text{otherwise} \end{cases} \tag{4.8}$$

So the keypoint is rejected if it is not an absolute maximum in magnitude of

response in its scale-space neighbourhood at a given scale. Hence instead of shifting the keypoints by Taylor Expansions, we keep them and remove parasitic keypoints with lower absolute response situated too close to the keypoints with stronger responses.

Therefore, at this step we need two parameters:

- the size n of the neighbourhood $\Omega(x_{dp})$ and
- the threshold th_{DoG} value

In our work we use $n = 5$ and the th_{DoG} is set to 0.01 and different values were also tested as it will be presented in results section.

The module uses 2D sliding window technique, to reduce the number of off-chip memory accesses and to make pipelining possible on the FPGA. The input of the module is: The given DoG image. The output of the module is: List of Candidate keypoints.

4.3.4 Edge detector module

This module checks if a candidate keypoint is situated on an edge or not. In the former case it is removed from a list of candidate points. Accordingly to the SIFT approach Hessian matrix (eq. 4.4) is calculated in the vicinity of the given point. The elements of the Hessian matrix are computed as follows, here we denote by DoG the array containing the DoG values $D(x, y)$, see eq. 4.3 of the given scale and interval:

$$\begin{aligned} D_{xx} &= DOG_{[1][2]} + DOG_{[1][0]} - 2 \times DOG_{[1][1]} \\ D_{yy} &= DOG_{[2][1]} + DOG_{[0][1]} - 2 \times DOG_{[1][1]} \\ D_{xy} &= \frac{DOG_{[2][2]} + DOG_{[2][0]} - DOG_{[0][2]} + DOG_{[0][0]}}{4} \end{aligned} \tag{4.9}$$

To store the required DoG elements around the candidate keypoint a 2D sliding window temporary memory is used in the module. If the determinant of the Hessian matrix is greater than 0, and the Hessian matrix trace sum squared is less than a predefined ϵ value the candidate keypoint still remains a candidate keypoint accordingly to the SIFT algorithm, see section 4.2.1. The ϵ value is calculated from the curvature threshold.

$$\epsilon = \frac{(curvature\ threshold + 1.0)^2}{curvature\ threshold} \times det(H) \tag{4.10}$$

In other case the candidate keypoint is removed from the keypoint list, as belonging to a border and not to a corner.

Inputs of this module are: DoG image at the given candidate keypoint, list of the candidate keypoints, the previous results. Outputs of this module are: filtered list of the candidate keypoints.

4.4 Experiments and results

In this section the implementations are compared in terms of precision of detection of keypoints, processing speed and dissipation power. The baseline for validation of different modules developed was the OpenSIFT implementation (Hess, 2010). The experiments were conducted on the "Grasping In The Wild (GITW) dataset (LaBRI, 2016)" freely available for research at NAKALA CNRS server [1]. The dataset is described it in the Chapter 3.3.2.

[1] https://www.labri.fr/projet/AIV/graspinginthewild.php

4.4.1 GFDG module validation

To validate the GFDG module we focus on the comparison of the Gaussian filtering implemented in FPGA with regard to the reference software. The GFDG module was compared to the OpenCV 2.4.13.7 cvSmooth function. This function is convolving an image with a Gaussian kernel and the output of the function is a Gaussian-convolved image. On the FPGA board OpenCV 4 has been used and the cvSmooth function renamed to GaussianBlur. This result and the module result are compared to each other and if the absolute difference between the two results is less than a predefined threshold ϵ_1 the module is validated. Hence for each frame $I_m, m = 1, .., M$ in our dataset the first computation is the the maximal value MD_m of the absolute pixel values difference between the two results dI_m:

$$MD_m = \max_{(x,y)}(|OCV_G(I_m(x,y)) - FPGA_G(I_m(x,y))|) \tag{4.11}$$

Here OCV_G is the OpenCV cvSmooth function, $FPGA_G$ is the GFDG module. Then the mean maximal absolute difference on the whole dataset MMD is computed:

$$MMD = \frac{1}{M}\sum_{m=1}^{M} MD_m \tag{4.12}$$

with M is the number of video frames in the whole dataset.

In the simulation of the Vivado HLS 2018.3 (Xilinx, 2018b) shows, the MMD between the GFDG module and the OpenCV 2.4.13.7 cvSmooth function are smaller than $\epsilon_1 = 1 \times 10^{-5}$ in the whole bowl dataset.

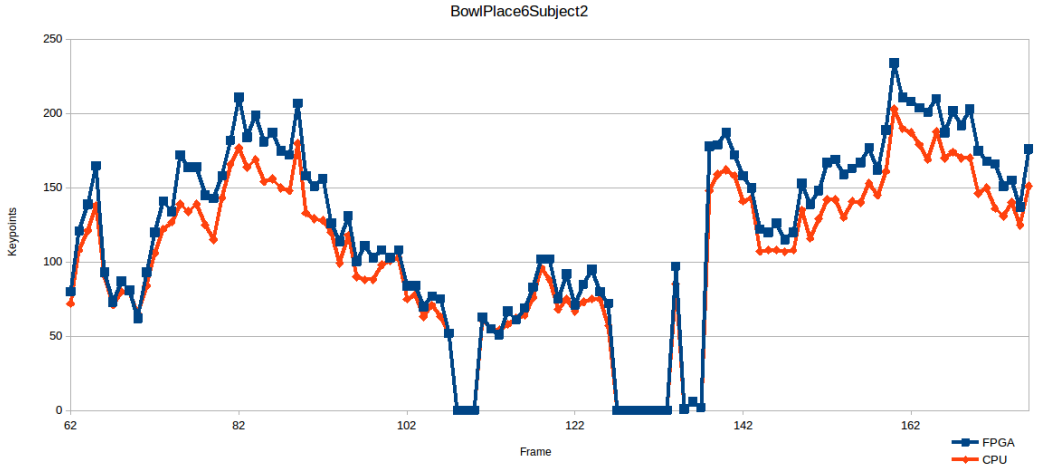


Figure 4.5: Comparison between the OpenSIFT (Hess, 2010) and the FPGA implementation detected keypoints. Y-axis is the number of the extracted keypoints, X-axis is the frame number. FPGA-detected keypoints are in blue, CPU-detected keypoints are in red.

Figure 4.6: Visual comparison of detected sets of keypoints. On the left: OpenSIFT (Hess, 2010) extracted keypoints. On the right: the FPGA result.

4.4.2 Comparison of the CPU implementations with FPGA implementations

To illustrate the behavior of our simplified SIFT detector implementation in FPGA and in comparison with OpenSIFT (Hess, 2010) we present the number of detected points on each frame of one sequence from our dataset in Figure 4.5 for both detectors. As we expected, the FPGA implementation detects more keypoints.

The video sequence in this experiment is the BowlPlace6Subject2 glass video.

During this experiment, the default parameters (Table 4.1) have been used. Figure 4.6 shows an example of recognizing a frame of a video for visual comparison with both the OpenSIFT (Hess, 2010) and FPGA implementations. One can see that the keypoint sets are quite similar. The difference is coming from the fact that Hess (2010) uses Taylor expansion in the keypoint localization step, and the FPGA implementation uses the Non-Maximum Suppression only, see section 4.3.3.

4.4.3 The FPGA keypoint detection module assessment

Here we compare the overall results of detection of SIFT keypoints by our FPGA solution with detection by the reference software OpenSIFT (Hess, 2010). We consider that two keypoints (FPGA-detected and OpenSIFT-detected) coincide if the Manhattan distance between them is less than a predefined threshold ϵ_2. As a threshold value, we have taken 2. Hence:

$$\begin{aligned} \text{True Positive:} \quad & x_{CPU} \exists x_{FPGA} : \quad \| \, x_{CPU} - x_{FPGA} \, \| < \epsilon_2 \\ \text{False negative:} \quad & x_{CPU} \cap (x_{FPGA} = \emptyset) \\ \text{False positive:} \quad & x_{FPGA} \cap (x_{CPU} = \emptyset) \\ \text{True negative:} \quad & \emptyset \quad \text{CPU} \quad \text{and} \quad \emptyset \quad \text{FPGA} \end{aligned} \tag{4.13}$$

where x_{CPU} is a keypoint in Hess (2010) and x_{FPGA} is a keypoint in the FPGA implementation. True positive (TP) is the case when the distance between the x_{CPU} and x_{FPGA} is less than a predefined threshold ϵ_2. False negative (FN) is when no x_{FPGA} points in a radius ϵ_2 circle in a center x_{CPU}. False positive (FP) is when no x_{CPU} points in a radius ϵ_2 circle in a center x_{FPGA}. True negative (TN) corresponds to the case when there is no keypoint in both of the implementations.

Precision (P) and recall (R) are calculated accordingly.

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN} \tag{4.14}$$

Table 4.1: The default parameter list in experiments.

name	**value**
image size	480px × 480px
octave	3
scale	6
Gaussian kernel size	5 × 5
initial σ	1.6
contrast threshold	0.04
curvature threshold	10
FPGA only parameters	
NMS kernel size	5 × 5
th_{DoG}	0.01

Table 4.2 illustrates the results when default parameters have been used, see Table 4.1. The first column contains the name of the video sequences. The second column depicts the average number of OpenSIFT (Hess, 2010)-extracted keypoints in the frames of a video sequence and the standard deviation. The average per/frame number of our FPGA-extracted keypoints and the standard deviation are given in the first column. The final two columns are the average precision and the average recall. It can be seen that the numbers of recall are quite high – up to 0.97 while precision is lower – from 0.77 up to 0.91. This confirms the general trend that FPGA based implementation with a simplified algorithm increases the number of supplementary detections.

Table 4.2: Comparison of the number of keypoints in case of OpenSIFT (CPU)(Hess, 2010) and our FPGA implementations. P is the Precision and R is the Recall.

Name	CPU	FPGA	P	R
BP1S1	37.08 ± 25.42	41.21 ± 27.51	0.79 ± 0.24	0.93 ± 0.11
BP1S2	41.66 ± 32.40	46.63 ± 34.39	0.77 ± 0.24	0.92 ± 0.12
BP1S3	29.02 ± 16.96	32.62 ± 18.03	0.81 ± 0.21	0.92 ± 0.15
BP1S4	48.61 ± 30.85	53.67 ± 32.95	0.83 ± 0.17	0.93 ± 0.08
BP4S1	13.99 ± 16.67	15.49 ± 17.62	0.85 ± 0.25	0.9 ± 0.21
BP4S2	11.21 ± 7.80	11.94 ± 8.25	0.86 ± 0.21	0.93 ± 0.16
BP4S3	38.54 ± 24.42	42.39 ± 26.54	0.86 ± 0.16	0.94 ± 0.08
BP4S4	3.92 ± 8.13	4.25 ± 9.33	0.89 ± 0.25	0.92 ± 0.23
BP5S1	17.43 ± 18.12	19.09 ± 18.92	0.83 ± 0.24	0.93 ± 0.14
BP5S2	9.27 ± 11.01	10.29 ± 11.93	0.83 ± 0.24	0.93 ± 0.15
BP5S3	20.59 ± 15.46	22.54 ± 16.91	0.86 ± 0.16	0.94 ± 0.11
BP6S1	73.51 ± 45.45	83.12 ± 51.17	0.85 ± 0.12	0.94 ± 0.06
BP6S2	105.92 ± 56.04	120.64 ± 65.22	0.85 ± 0.07	0.93 ± 0.06
BP6S3	77.79 ± 48.74	88.50 ± 55.44	0.84 ± 0.16	0.95 ± 0.06
BP6S4	22.70 ± 46.15	26.05 ± 52.62	0.77 ± 0.23	0.92 ± 0.11
BP7S1	2.48 ± 4.22	2.61 ± 4.45	0.91 ± 0.22	0.91 ± 0.22
BP7S2	4.38 ± 7.14	5.00 ± 8.04	0.84 ± 0.27	0.95 ± 0.14
BP7S3	4.55 ± 8.22	5.10 ± 8.99	0.84 ± 0.29	0.97 ± 0.09

Different parameters have been tested too. The th_{DoG} has been set to 0, 0.008, 0.009, 0.01, 0.011, 0.012. In those cases, the recall and the precision were lower then in the experiment with 0.01 value. When the initial scale parameter σ was set to 0.45 the average number of extracted keypoints was higher (OpenSIFT (Hess, 2010) 167.73, in our FPGA implementation 102.60). The distances between the keypoints are small, and the NMS removed them. That is why the OpenSIFT (Hess, 2010) extracted more keypoints than our FPGA implementation in that case. The aver-

age recall was 0.58, which is lower when the initial σ was set to 1.6. The average precision was 0.88, which is the same as when initial σ is set to 1.6.

Now we trace the behaviour of our FPGA implementation in different octaves. In case when the initial σ was set to 1.6 the average number of extracted keypoints in the 1^{st} octave was 15.93 with OpenSIFT (Hess, 2010) and 18.83 with our FPGA implementation on the whole bowl subset.

For the second octave, OpenSIFT (Hess, 2010) has extracted an average number of 11.72 keypoints and our FPGA implementation 12.28 respectively on the whole bowl subset as well. There are some cases when in the second octave more keypoints have been found than in the first octave, like BowlPlace1Subject2 and BowlPlace6Subject1. However, the extracted keypoints number decreased in the 3^{rd} octave (OpenSIFT (Hess, 2010) 3.61 keypoints in average and our FPGA implementation 3.96 keypoints in average respectively) compared to the 1^{st} and 2^{nd} octaves. Thus, the behaviour of our SIFT detector implemented in FPGA is generally the same: it gives more points than reference OpenSIFT implementation in different octaves.

These results showed that our FPGA and the full-software OpenSIFT (Hess, 2010) keypoints are almost equivalent. In that case, equivalent means that the keypoints in the reference images are really close to each other and situated at a distance less than the predefined threshold of 2 pixels.

Test of different NMS thresholds

In this subsubsection it will be shown why we decided to set the NMS threshold to 0.01.

Table 4.3 shown the average recall of the different NMS threshold. It can be

Table 4.3: The average recall in case of different NMS threshold values. The red column is when the NMS threshold was set to 0.01. The average recall is higher than 0.9 when the threshold is 0.01.

	NMS threshold						
R	**0**	**0.008**	**0.009**	**0.01**	**0.011**	**0.012**	**0.02**
BP1S1	0.93 ± 0.06	0.93 ± 0.06	0.93 ± 0.06	0.92 ± 0.07	0.73 ± 0.13	0.57 ± 0.15	0.1 ± 0.08
BP1S2	0.95 ± 0.06	0.95 ± 0.06	0.95 ± 0.06	0.91 ± 0.12	0.74 ± 0.15	0.6 ± 0.19	0.13 ± 0.09
BP1S3	0.94 ± 0.07	0.94 ± 0.07	0.94 ± 0.07	0.91 ± 0.14	0.66 ± 0.18	0.46 ± 0.21	0.04 ± 0.05
BP1S4	0.96 ± 0.06	0.96 ± 0.06	0.96 ± 0.06	0.94 ± 0.07	0.71 ± 0.14	0.55 ± 0.17	0.11 ± 0.09
BP4S1	0.98 ± 0.06	0.98 ± 0.06	0.98 ± 0.06	0.93 ± 0.16	0.74 ± 0.22	0.6 ± 0.26	0.2 ± 0.28
BP4S2	0.97 ± 0.08	0.97 ± 0.08	0.97 ± 0.08	0.93 ± 0.13	0.55 ± 0.26	0.34 ± 0.28	0.1 ± 0.19
BP4S3	0.97 ± 0.07	0.97 ± 0.07	0.97 ± 0.07	0.93 ± 0.09	0.64 ± 0.23	0.45 ± 0.22	0.08 ± 0.1
BP4S4	0.94 ± 0.21	0.94 ± 0.21	0.94 ± 0.21	0.9 ± 0.22	0.57 ± 0.39	0.38 ± 0.39	0.08 ± 0.21
BP5S1	0.96 ± 0.09	0.96 ± 0.09	0.96 ± 0.09	0.94 ± 0.11	0.73 ± 0.24	0.58 ± 0.27	0.12 ± 0.19
BP5S2	0.96 ± 0.08	0.96 ± 0.08	0.96 ± 0.08	0.93 ± 0.16	0.72 ± 0.26	0.57 ± 0.28	0.07 ± 0.11
BP5S3	0.97 ± 0.11	0.97 ± 0.11	0.97 ± 0.11	0.93 ± 0.13	0.64 ± 0.23	0.47 ± 0.24	0.07 ± 0.13
BP6S1	0.95 ± 0.06	0.95 ± 0.06	0.95 ± 0.06	0.93 ± 0.08	0.81 ± 0.11	0.71 ± 0.13	0.22 ± 0.19
BP6S2	0.94 ± 0.03	0.94 ± 0.03	0.94 ± 0.03	0.93 ± 0.04	0.79 ± 0.08	0.66 ± 0.1	0.14 ± 0.1
BP6S3	0.96 ± 0.04	0.96 ± 0.04	0.96 ± 0.04	0.95 ± 0.04	0.8 ± 0.15	0.7 ± 0.17	0.2 ± 0.12
BP6S4	0.96 ± 0.05	0.96 ± 0.05	0.96 ± 0.05	0.95 ± 0.06	0.71 ± 0.24	0.57 ± 0.25	0.09 ± 0.1
BP7S1	0.96 ± 0.11	0.96 ± 0.11	0.96 ± 0.11	0.87 ± 0.29	0.66 ± 0.38	0.52 ± 0.4	0.11 ± 0.19
BP7S2	0.97 ± 0.15	0.97 ± 0.15	0.97 ± 0.15	0.96 ± 0.15	0.67 ± 0.36	0.44 ± 0.34	0.05 ± 0.11
BP7S3	0.97 ± 0.09	0.97 ± 0.09	0.97 ± 0.09	0.95 ± 0.11	0.71 ± 0.36	0.57 ± 0.37	0.11 ± 0.21
Total	0.96 ± 0.08	0.96 ± 0.08	0.96 ± 0.08	0.93 ± 0.12	0.7 ± 0.23	0.54 ± 0.25	0.11 ± 0.14

Table 4.4: The average precision in case of the different NMS threshold values. The red column is when the NMS threshold was set to 0.01. The average precision is higher than 0.81 when the threshold 0.01.

	NMS threshold						
P	**0**	**0.008**	**0.009**	**0.01**	**0.011**	**0.012**	**0.02**
BP1S1	0.3 ± 0.12	0.54 ± 0.13	0.66 ± 0.12	0.8 ± 0.1	0.8 ± 0.11	0.81 ± 0.11	0.81 ± 0.27
BP1S2	0.25 ± 0.1	0.49 ± 0.13	0.63 ± 0.13	0.77 ± 0.14	0.77 ± 0.16	0.76 ± 0.19	0.76 ± 0.28
BP1S3	0.33 ± 0.13	0.54 ± 0.14	0.64 ± 0.13	0.8 ± 0.09	0.8 ± 0.11	0.78 ± 0.14	0.79 ± 0.34
BP1S4	0.24 ± 0.09	0.49 ± 0.12	0.63 ± 0.11	0.8 ± 0.12	0.8 ± 0.13	0.8 ± 0.13	0.81 ± 0.26
BP4S1	0.18 ± 0.12	0.42 ± 0.19	0.57 ± 0.17	0.79 ± 0.21	0.8 ± 0.23	0.81 ± 0.21	0.83 ± 0.33
BP4S2	0.11 ± 0.08	0.34 ± 0.25	0.54 ± 0.25	0.8 ± 0.24	0.82 ± 0.28	0.82 ± 0.34	0.81 ± 0.37
BP4S3	0.14 ± 0.08	0.37 ± 0.17	0.55 ± 0.21	0.78 ± 0.17	0.78 ± 0.19	0.78 ± 0.22	0.83 ± 0.3
BP4S4	0.05 ± 0.09	0.33 ± 0.31	0.57 ± 0.36	0.89 ± 0.24	0.86 ± 0.27	0.9 ± 0.23	0.93 ± 0.16
BP5S1	0.13 ± 0.09	0.44 ± 0.17	0.63 ± 0.21	0.79 ± 0.18	0.79 ± 0.21	0.79 ± 0.25	0.71 ± 0.39
BP5S2	0.12 ± 0.11	0.41 ± 0.25	0.58 ± 0.28	0.82 ± 0.19	0.82 ± 0.2	0.83 ± 0.22	0.86 ± 0.29
BP5S3	0.14 ± 0.09	0.42 ± 0.18	0.58 ± 0.2	0.8 ± 0.14	0.81 ± 0.16	0.81 ± 0.17	0.8 ± 0.39
BP6S1	0.29 ± 0.11	0.54 ± 0.15	0.67 ± 0.14	0.78 ± 0.15	0.78 ± 0.15	0.78 ± 0.15	0.8 ± 0.19
BP6S2	0.36 ± 0.16	0.55 ± 0.17	0.65 ± 0.15	0.79 ± 0.06	0.79 ± 0.06	0.79 ± 0.06	0.8 ± 0.14
BP6S3	0.33 ± 0.14	0.54 ± 0.12	0.66 ± 0.11	0.79 ± 0.1	0.79 ± 0.12	0.79 ± 0.1	0.78 ± 0.17
BP6S4	0.11 ± 0.16	0.41 ± 0.23	0.55 ± 0.25	0.79 ± 0.17	0.77 ± 0.17	0.77 ± 0.17	0.8 ± 0.25
BP7S1	0.06 ± 0.09	0.28 ± 0.26	0.53 ± 0.3	0.84 ± 0.26	0.88 ± 0.2	0.87 ± 0.24	0.79 ± 0.38
BP7S2	0.09 ± 0.13	0.33 ± 0.35	0.57 ± 0.36	0.81 ± 0.27	0.82 ± 0.28	0.85 ± 0.25	0.91 ± 0.19
BP7S3	0.1 ± 0.12	0.48 ± 0.29	0.65 ± 0.32	0.85 ± 0.24	0.86 ± 0.26	0.87 ± 0.26	0.93 ± 0.26
Total	0.18 ± 0.11	0.44 ± 0.2	0.6 ± 0.21	0.81 ± 0.17	0.81 ± 0.18	0.81 ± 0.19	0.82 ± 0.27

Table 4.5: The average number of KPs extracted in case of different NMS threshold values. The red column shows when the NMS threshold is set to 0.01. The number of extracted KPs are computed just using the first octave of the Gaussian pyramids.

	NMS threshold							
# of KP	**CPU**	**0**	**0.008**	**0.009**	**0.01**	**0.011**	**0.012**	**0.02**
BP1S1	24.1 ± 10.4	76.7 ± 26	40.8 ± 15.8	33.5 ± 13.4	27.4 ± 11.9	22.2 ± 10.2	17.7 ± 8.8	3 ± 2.4
BP1S2	20.3 ± 12	73.4 ± 28	37.4 ± 18	29.5 ± 16.1	23.7 ± 13.6	19.7 ± 11.9	16.1 ± 10.1	3.5 ± 2.9
BP1S3	17.3 ± 8.8	51.8 ± 24.2	29.4 ± 11.9	25.1 ± 11.5	20 ± 10	14.8 ± 8.6	10.7 ± 7.2	0.9 ± 1
BP1S4	20.9 ± 12.6	86.3 ± 42.6	40.1 ± 21	31.4 ± 17.5	24.4 ± 14.7	19.1 ± 12.5	15 ± 10.4	3.2 ± 3
BP4S1	8.6 ± 12.2	36.5 ± 23.5	16.6 ± 17.4	13.1 ± 15.1	10.1 ± 13.4	8.1 ± 12	6.7 ± 10.8	1.7 ± 3.9
BP4S2	4 ± 2.7	41.8 ± 25.8	13.9 ± 9.9	7.6 ± 5.1	4.6 ± 2.9	2.7 ± 1.8	1.5 ± 1.2	0.4 ± 0.7
BP4S3	12.4 ± 11.1	74.2 ± 39.3	29.1 ± 22.4	20.6 ± 16.6	14.8 ± 13	10.4 ± 9.6	7.7 ± 7.5	1.6 ± 2.4
BP4S4	1.6 ± 3.3	24.8 ± 22.9	4.4 ± 6.8	2.7 ± 5.2	1.7 ± 3.8	1.2 ± 2.9	0.8 ± 2.3	0.2 ± 0.7
BP5S1	6.9 ± 6.8	44.2 ± 18.6	13.9 ± 11.7	10.4 ± 9.6	8.1 ± 7.9	6.3 ± 6.4	5.3 ± 5.8	1.2 ± 1.7
BP5S2	3.7 ± 4.6	23.6 ± 20.6	7.3 ± 7.9	5.5 ± 6.4	4.2 ± 5.2	3.4 ± 4.3	2.7 ± 3.6	0.4 ± 0.7
BP5S3	8.2 ± 6.7	52.9 ± 23	18 ± 11.8	13.2 ± 9.6	9.8 ± 7.9	7.1 ± 6.2	5.5 ± 5.2	0.6 ± 1
BP6S1	32.9 ± 22.2	97.1 ± 55.3	55.7 ± 37	46.1 ± 31.1	39.1 ± 26.1	33.7 ± 21.9	29 ± 19.2	7.8 ± 5.8
BP6S2	64.3 ± 35.3	150.6 ± 66.5	101.6 ± 51.9	88.7 ± 47.2	76.7 ± 42.6	65.6 ± 37.2	55.2 ± 31.8	10.8 ± 7.8
BP6S3	40.7 ± 25.5	103.6 ± 46.3	66.1 ± 36.9	56.7 ± 34.2	49.5 ± 31.7	43.8 ± 28.7	38.6 ± 25	10.1 ± 7.2
BP6S4	12.8 ± 26	39.9 ± 58.8	21.9 ± 41.4	18.2 ± 36.1	15.2 ± 31.1	12.7 ± 26.6	10.5 ± 22.2	2.1 ± 5.4
BP7S1	1.3 ± 2	14.6 ± 12	4.1 ± 4.8	2.4 ± 3.2	1.4 ± 2.3	1.1 ± 1.9	0.9 ± 1.6	0.2 ± 0.6
BP7S2	2.1 ± 3.4	17.3 ± 16.7	5.3 ± 7	3.4 ± 5.1	2.5 ± 4	1.9 ± 3.2	1.5 ± 2.7	0.2 ± 0.6
BP7S3	2.6 ± 4.8	16 ± 20.1	4.7 ± 7.7	3.7 ± 6.6	3 ± 5.3	2.3 ± 4.3	1.8 ± 3.4	0.3 ± 0.6
Total	15.8 ± 11.7	57 ± 31.7	28.4 ± 19	22.9 ± 16.1	18.7 ± 13.7	15.3 ± 11.7	12.6 ± 9.9	2.7 ± 2.7

seen the average recall is higher when the NMS threshold is set to less than 0.011. Table 4.5 indicates, the number of extracted keypoints is higher when the NMS threshold set to low value (0, 0.008, 0.009, 0.01). The rate of the false negative keypoints are less when the NMS threshold value is set between 0 and 0.01. However, if the NMS threshold set too low there are too many false positives extracted KP as Table 4.4 shown. The precision is low when the NMS threshold set to 0, 0.08, 0.09. The precision is higher when the NMS threshold was set to 0.01, 0.011, 0.012, and 0.02. Thus, the NMS threshold set to 0.01 was selected as the default value because the recall and precision ratio was the highest to compare to the other NMS thresholds.

4.4.4 Used resources

The resource usage of the complete system is presented in Table 4.6. The implementation is based on the Vivado 2018.3 (Xilinx, 2018b) software, which can report the

Table 4.6: 1 octave resource usages on Xilinx ZCU102 FPGA Board from Vivado 2018.3 (Xilinx, 2018b) when the default parameters in Table 4.1 has been used.

Resource #	Full System	SIFT computation module	Available
LUT	143,667	117,620	274,080
FF	189,944	157,946	548,160
BRAM	461.5	461.5	912
DSP	938	938	2,520

Table 4.7: Resource usage estimation of the main modules of the system based on the Vivado HLS report.

Resources #	Gauss filter and Difference of Gaussian (GFDG)	scale-space extrema (SSE)	Non-Maximum Suppression (NMS)	Edge Detection (ED)	Total
FF	17810	5569	9314	6630	219243
BRAM	15	6	6	3	1011
DSPs	127	0	0	44	938
LUT	17572	7255	19026	6160	276366

hardware requirements and clock frequency of the circuit. Here, the default parameters presented in Table 4.1 have been used. The results show that around 50 % of the resources have been used on the Xilinx ZCU 102 FPGA board. Therefore, there are enough resources to implement other steps of the prosthetic arm controlling algorithm.

The system is implemented in Vivado HLS as a single IP block. The Vivado toolchain does not hold the details of this block's exact inner structure during the implementation step. Therefore, the area requirements of the submodules are summarized using the Vivado HLS area report as shown in Table 4.7. The complete system is build up using six GFDG modules, four SSE modules, four NMS modules

and four ED modules. The most resource-consuming part is the GFDG computation module, where the majority of the BRAM and DSP slices are used. The area requirements of the SSE and NMS units are dominated by the logic resources (FFs and LUTs).

4.4.5 Discussion

In this section, we compare our method with other platforms and previous solutions.

Comparison of FPGA implementation with other platforms

Table 4.8 shows how many frames can be processed on the given hardware per second. Our FPGA solution is reported in the column 1 (Xilinx UltraScale+ ZCU102). The result shows FPGAs, Intel Xeon E5-2620 server CPU, GPU, and the CMOS Vision Sensor can process more than 70 frames per second. Our FPGA implementation used the default parameters in Table 4.1. The Intel Xeon E5-2620 server CPU and the ARM Cortex-A53 were measured with the OpenSIFT (Hess, 2010) SIFT implementation with the same parameters as our FPGA implementation. The analog solution (CMOS Vision Sensor) computed just the first step of the SIFT algorithm, the Gaussian pyramid calculation. However, energy consumption of the Intel Xeon E5-2620 server CPU, NVIDIA Jetson TX 2, and the NVIDIA GeForce GTX 580 GPU are higher than the FPGAs. The CMOS Vision Sensor solution only handles 176×120 pixels images, which is much smaller than the FPGAs.

The power dissipation of the proposed SIFT module is relatively low, just 5.6W. The CMOS Vision Sensor has the lowest energy consumption 70mW, however, the input resolution (176px $\times$ 120px) and the numbers of pixels being processed (2.6 million px) are lower than in the FPGAs.

Comparison of the power consumption shows that the Intel Xeon E5-2620 server

Table 4.8: Comparison of SIFT keypoint extraction in different platforms. The Intel Xeon E5-2620 and ARM Cortex-A53 are used the OpenSIFT(Hess, 2010) to extract the keypoint. The Xilinx UltraScale+ ZCU102 are used our FPGA implementation to extract the keypoint. All three platform which used OpenSIFT are used with the default parameters (Table 4.1). On the CMOS Vision Sensor, only the time for Gaussian Pyramid calculation is reported. In the case of NVIDIA Jetson TX 2 the computation time is based on the report of da Costa Barreiros (2020).

	Xilinx UltraScale+ ZCU102 Our solution	**Intel Xeon E5-2620**	**ARM Cortex-A53**	**CMOS Vision Sensor**	**NVIDIA GeForce GTX 580**	**NVIDIA Jetson TX2 2**
implemented algorithm	OpenSIFT (FPGA version)	OpenSIFT (Hess, 2010)	OpenSIFT (Hess, 2010)	Gaussian Pyramid (Rodríguez-Vázquez et al., 2009)	CudaSift (Björkman et al., 2014)	da Costa Barreiros (2020)
resolution (px)	480 x 480	480 x 480	480 x 480	176 x 120	1280 x 960	**2560 x 1920**
frames / second	**135**	21	5	125	78.74	1.45
dissipated power	5.6W	80W	2.5W	**70mW**	244W	7W-15W

CPU, and that the GPU, are using too much energy for a wearable device. In contrast, the power consumption of the FPGA, the analog CMOS Vision Sensor, the NVIDIA Jetson TX 2, and the ARM Cortex-A53 CPU, are sufficiently low for our application. However, the processing speed of the ARM Cortex-A53 CPU and the NVIDIA Jetson TX2 is too low and does not fulfil our demands for real-time processing. The Intel Xeon E5-2620 CPU, FPGA, NVIDIA GTX 580 GPU, and CMOS Vision sensor are capable of real-time processing. The analog solutions can only process small input images, whereas the FPGA, GPU, and both of the CPUs can handle sufficiently high image resolution which is higher than Full HD. The FPGA solution is equivalent to the full software implementation OpenSIFT (Hess, 2010) in terms of proximity and number of detected keypoints.

The disadvantage of the FPGA solution is that the input image size is small 480px × 480px, due to the limited 32.1Mb internal memory of the Xilinx UltraScale+ ZCU102. The CPUs and the GPU's solutions can process higher-resolution input. Nevertheless, for controlling the robotic arm, this resolution is sufficient.

Comparison of the power consumption, processing speed and input image size for the different architectures shows that the FPGA is the best choice for creating a wearable device, as Table 4.8 shows.

The latency of the system is 238390 clock cycles for the input image size of 480px × 480px this is based on the Vivado HLS 2018.3 report (Xilinx, 2018b). The current implementation is running on 200MHz clock frequency according to the post routing timing analysis in Vivado 2018.3 (Xilinx, 2018b), so it can process 135 full 480px × 480px images per second, which is higher than the current input video frame rate. Therefore, a real-time processing is achievable.

Comparison of the different FPGA implementations

In this part, we compared our implementation to the state-of-the-art FPGA solutions. As Table 4.9 indicated, we used a different FPGA board (Xilinx UltraScale+ ZCU102) from that used in previous studies. Ginés et al. (2020) and Shao et al. (2015) use Xilinx Virtex-5. Pablo et al. (2018) implementation run on Zedboard, and Vourvoulakis et al. (2017, 2016) developed their solution on Intel DE2i-150. Chang et al. (2013) developed their solution for a Xilinx Virtex II Pro. Our solution can process a 480px × 480px resolution image which has a higher resolution compared to the Chang et al. (2013) and Shao et al. (2015), but Ginés et al. (2020) and Vourvoulakis et al. (2017, 2016) implementations can process images with higher resolution of 640px × 480px. Our proposed architecture can compute 135 frames per second, which is higher than both Ginés et al. (2020) and Vourvoulakis et al. (2017, 2016) and Shao et al. (2015) In comparison, Chang et al. (2013) solution can process 900 frames per second, however, it uses lower image resolution. Our implementation is a software/hardware solution similarly to Pablo et al. (2018), and Chang et al. (2013), and Shao et al. (2015). Other papers such as Ginés et al. (2020) and Vourvoulakis et al. (2017, 2016) use solutions that are a hardware but simplified implementation of the SIFT, which impacts the algorithm accuracy. Our solution is using a float number representation instead of the fixed-point representation. This

causes a higher resource demand, but it also provides better accuracy.

Table 4.9: Comparison of our work with different FPGA implementations.

	Our implementation	**Doménech-Asensi**	**Rubio-Ibáñez**	**Vourvoulakis**	**Chang**	**Shao**
board	ZCU102	Xilinx Virtex-5	ZedBoard	Intel DE2i-150	Xilinx Virtex II Pro	Xilinx Virtex-5
resolution (px)	480 x 480	**640 x 480**	-	**640 x 480**	320 x 240	292 x 520
frames/second	135	99	-	81	**900**	38
architecture	HW/SW	HW	HW/SW	HW	HW/SW	HW/SW
number representation	float	fixed-point	fixed-point	fixed-point	fixed-point	fixed-point

4.5 Conclusion

We created a 32 bit floating point SIFT keypoint extractor hybrid solution, which is running on FPGA and can generate the same results as the software implementation of OpenSIFT (Fejér et al., 2021a).

We have made some simplification like Non-Maxima Suppression, which is not changing the accuracy of the original algorithm (Fejér et al., 2021a).

Computing time of our implementation is 7.407ms (135fps), which is fast enough for real-time computing (Fejér et al., 2021a).

The total power consumption of the proposed system is 5.6W, which is suitable for wearable devices (Fejér et al., 2021a).

In the next Chapter 5 we will focus on the Gaze-Driven CNN (González-Díaz et al., 2019) preprocessing algorithms and discuss the possible acceleration steps.

Chapter 5

Optimized implementations of preprocessing steps

5.1 Introduction

This chapter is about the preprocessing steps of the Gaze Driven CNN. The preprocessing is necessary to improve the accuracy and the computational speed of the algorithm. For the object recognition and localization the sources of the most annoying noise are: i) the scattered eye-movement, in which case the gaze fixation is not on the object to grasp, ii) the head movement of the user, which also affects the localization of the gaze point. There are two steps of the preprocessing: the gaze alignment step and the gaze point noise reduction. We present the gaze alignment in the subchapter 5.2. The gaze point noise reduction is presented in subchapter 5.3.

The gaze alignment module is aimed to register the gaze points from consecutive frames to the reference frame's plane. To achieve that, first, it is necessary to extract some features from the reference frame and the other frames. In our work, we proposed the usage of SIFT (Lowe, 2004) keypoint key points and described its hybrid implementation in Chapter 4. In the subchapter 5.2 we will present the usage keypoint in matching frames. The keypoint descriptors of the reference frame and

the other frames are compared in the FLANN matcher (Muja et Lowe, 2009), which is described in the subchapter 5.2.1. The homography estimation with RANSAC iteration (Fischler et Bolles, 1981) is made on the paired SIFT (Lowe, 2004) key-points to find the homography matrix which is able to transform the other frame gaze point to the reference frame's plane. The homography estimation is described in subchapter 5.2.2 and the RANSAC algorithm is described in the subchapter 5.2.

When all the gaze points are in the reference frame's plane, the gaze point noise reduction module removes the outlier gaze points. DBSCAN (Ester et al., 1996) is used to cluster the gaze points, this can remove the noisy gaze points which are caused by the scattering and the head movements of the user. The DBSCAN (Ester et al., 1996) algorithm is described in subchapter 5.3.1. The biggest cluster of gaze points is used to estimate the position of the gaze points with the KDE (Pedregosa et al., 2011). The KDE (Pedregosa et al., 2011) algorithm is described in subchapter 5.3.2.

5.2 Gaze point alignment

Tobii's glass camera and eye-tracking system provides the coordinates of the gaze fixations in each video frame of the first-person integrated camera.

Even if the subject is looking at the same object to grasp during the object reaching, the projected gaze points will vary between two consecutive frames because of the body and ocular movements. Furthermore, saccades provoked by distractors can deviate from the fixation on the object. Hence, the first step consists in the estimating a gaze fixation in the current video frame using all the gaze fixations from the past frames. It is necessary to estimate and compensate the ego-motion between the past frames and the current frame to collect all gaze points in the same

frame. We show an illustration of such a collection in Figure 5.1, where the lightest is the gaze fixation point in the current frame, and the more distanced are the gaze points from the current timestamp, the darker they are.

Figure 5.1: Example of bowl place 4 subject 2 gaze point alignment. The points are the gaze points.

Motion compensation from the past frames to the current frame is realized by a sequential homography transformation computed between consecutive frames.

Suppose a video sequence given with N frames and a list of gaze points, $g_n = \{(g_{xn}, g_{yn}), n = 1...N\}$. Note that generally with the video frame rate and eye-tracker acquisition rate we have, only one gaze point recorded for each frame. Hence, for simplicity in our notations we put one gaze point by frame, without loosing generality of the method. The system operates as follows: for each pair of consecutive frames, it extracts the characteristic keypoints and local features. In our case, the keypoint extractor is the Scale-Invariant Feature Transform (SIFT) (Lowe, 2004) (SIFT). To compute the geometric transformation of frames, the key points have to be matched. Matching of SIFT points is a well studied problem. Therefore, we take an already

available solution. A Fast Library for Approximate Nearest Neighbours (FLANN)-based matcher (Muja et Lowe, 2009) is used to find the good matches between the SIFT descriptors of the two frames.

The final step is to estimate the homography transformation matrices, $H_n, n = 1, .., N$, with N, the number of the current frame, based on the good matches. Then, the gaze fixations can be projected from all frames into the current frame by a composition of homographies H_n. In this projection, we use a sliding window of duration, $\Delta t = 10$, frames which correspond with 400 ms time interval, with the scene apprehension time by the subjects in our experiments. Therefore, for the current frame, N, the collected gaze points are $\hat{g}_{N,n}, n = N - \Delta t, .., N$.

5.2.1 Fast Library for Approximate Nearest Neighbors-based matching

Fast Library for Approximate Nearest Neighbours (FLANN) (Muja et Lowe, 2009) is an image matching algorithm for fast approximate nearest neighbour searches in high dimensional spaces. These methods project the high-dimensional features to a lower-dimensional space. After that, the compact binary code is generated. Benefiting from the produced binary code, a fast image search can be performed via binary pattern matching or Hamming distance measurement, dramatically reducing the computational cost and further optimizing the efficiency of the search.

There are several implementations of the FLANN algorithm in OpenCV (Bradski, 2000).

- FLANN_INDEX_LINEAR = 0
- FLANN_INDEX_KDTREE = 1
- FLANN_INDEX_KMEANS = 2

- FLANN_INDEX_COMPOSITE = 3
- FLANN_INDEX_KDTREE_SINGLE = 4
- FLANN_INDEX_HIERARCHICAL = 5
- FLANN_INDEX_LSH = 6
- FLANN_INDEX_SAVED = 254
- FLANN_INDEX_AUTOTUNED = 255

In this research, the randomized kd-tree algorithm has been used. The traditional kd-tree technique (Friedman et al., 1977) performs well in low dimensions, but rapidly worsens in high dimensions. In our research, to achieve the real-time processing speed, it becomes required to settle for an approximative nearest neighbor in order to gain a speed-up over linear search. The algorithm may not always yield the precise nearest neighbors, but this speeds up the search process. So, nearest neighbors algorithm can cause accuracy loss, but their processing speed is faster than a brute force method.

Silpa-Anan et Hartley (2008) proposed an improved version of the kd-tree algorithm in which multiple randomized kd-trees are created. The original kd-tree approach divides the data in half on the dimension with the highest variance at each level of the tree. The split dimension for the randomized trees, in contrast, is randomly selected from the top D dimensions on which the data has the highest variance. Muja et Lowe (2009) uses the fixed value D = 5 (it is called trees in OpenCV (Bradski, 2000)). D = 5 performs well in Muja et Lowe (2009) datasets and also in GITW (LaBRI, 2016) datasets and does not significantly benefit from further adjustment. So we decided to use trees = 5 parameter. A single priority queue is kept across all the randomized trees when searching them, allowing the search to

be sorted by increasing distance from each bin boundary. A fixed number of leaf nodes are examined to determine the degree of approximation, at which time the search is stopped, and the top candidates are returned. In the Muja et Lowe (2009) approach, the user simply selects the level of search precision they want, which is utilized to determine how many leaf nodes will be investigated to obtain this level of precision during training.

In the OpenCV (Bradski, 2000) implementations of FLANN, there is one more other parameter, the checks, which specifies the number of times the trees in the index should be recursively traversed. Higher values give better precision, but also take more time. In our case, we used 50 after some preliminary tests.

To filter out the outliers, Lowe (2004) proposed a distance ratio test, which eliminates bad matches. A good match is found when the distance ratio between two closest matches of a keypoint under consideration is less than a predetermined value. For example, the two descriptors are n and m and m.distance $<$ threshold * n.distance. In our case, this threshold is 0.8.

5.2.2 Homography estimation

Homography matrix (Szeliski, 2011) is used to project a pixel from one image to another image. The homography relates to the transformation between two planes. Besides, it is a linear transformation, because the coordinates vector is multiplied by a transformation matrix.

The formula of the homography estimation is the following:

$$s \times \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{32} & h_{32} & h_{33} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.1}$$

where H is the transformation (homography) matrix and $(x, y, 1)$ are the homogeneous pixel coordinates which will be transformed to a new plane $(x', y', 1)$. The H matrix is a 3x3 matrix, but with 8 degrees of freedoms, as it is estimated up to a scale. The homography matrix is normalized with $h_{33} = 1$ or $h_{11}^2 + h_{12}^2 + h_{13}^2 + h_{21}^2 + h_{22}^2 + h_{23}^2 + h_{31}^2 + h_{32}^2 + h_{33}^2 = 1$.

The homography estimation is good to create panoramas, 3D reconstruction, and preprocess images for OCR and self-driving cars, find a predefined object.

To calculate a proper homography matrix, the outliers need to be eliminated. It can be done with the RANSAC iteration (Fischler et Bolles, 1981) or the Least-Median robust method (Rousseeuw, 1984) or the PROSAC-based robust method (Chum et Matas, 2005).

5.2.3 RANSAC algorithm

The RANdom SAmple Consensus (RANSAC) algorithm (Fischler et Bolles, 1981) is a general parameter estimation approach designed to cope with a large proportion of outliers in the input data. Unlike many of the common robust estimation techniques such as M-estimators and least-median squares that have been adopted by the computer vision community from the statistics literature, RANSAC was developed by the computer vision community.

RANSAC is a re-sampling technique that generates candidate solutions by using the minimum number of observations (data points) required to estimate the underlying model parameters. Unlike conventional sampling techniques that use as much of the data as possible to obtain an initial solution and then proceed to prune outliers, RANSAC uses the smallest set possible and proceeds to enlarge this set with consistent data points (Fischler et Bolles, 1981). The basic algorithm is summarized as follows:

1. Select randomly the minimum number of points required to determine the model parameters.

2. Solve for the parameters of the model.

3. Determine how many points from the set of all points fit to a predefined tolerance τ.

4. If the fraction of the number of inliers over the total number of points in the set exceeds a predefined threshold τ, re-estimate the model parameters using all the identified inliers and terminate.

5. Otherwise, repeat steps 1 through 4 (maximum of N times).

The number of iterations, N, is chosen high enough to ensure that probability p (usually set to 0.99) that at least one of the sets of random samples does not include an outlier. Let u represent the probability that any selected data point is an inlier and $v = 1 - u$ the probability of observing an outlier. K iterations of the minimum number of points denoted k are required, where $1 - p = (1 - u^k)^K$ and thus we can deduce that $K = \frac{log(1-p)}{log(1-(1-v)^k)}$

Thus, after matching the key-points in the two consecutive frames and estimating homography we can successively project by the composition of at most $N - m - 1$ homographies with N the number of the current frame and $N - m$ the number of the reference frame, the gaze points from previous frames into the current frame.

5.3 Gaze point noise reduction

The goal of this module is to reduce the noise of the gaze fixations projected into the current frame. It has two steps: the DBSCAN (Ester et al., 1996) clustering (see Chapter 5.3.1) and the KDE (Pedregosa et al., 2011) (see Chapter 5.3.2).

The DBSCAN (Ester et al., 1996) removes the outlier gaze points, which are caused by the saccades to the distractors. The Kernel density Estimator (KDE) (Pedregosa et al., 2011) is applied to estimate the smoothed gaze point position in the current frame.

5.3.1 Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (Ester et al., 1996) is a non-parametric algorithm, so the target number of the clusters is not predefined, but obtained accordingly to the density of the data and parameter settings of the algorithm. It is clustering a given set of points in some representation space. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) groups together points that are closely packed together and marks, as outliers, points that lie alone in low-density regions.

The DBSCAN has two parameters: the minimum number of points or samples (minPts), and the ϵ.

- The minPts is the number of samples (or total weight) in a neighbourhood for a point to be considered as a core point. This includes the point itself.
- The ϵ specifies how close points should be to each other to be considered a part of a cluster. Thus, if the distance between two points is less or equal to this value ϵ, these points are considered neighbours.

Figure 5.2 shows how the clustering is done. In this example the minPts=3 so there are 3 core points (squares), as in their ϵ-neighbourhood at least three points can be found. The three circles (core points and their neighbours) create a cluster, and the noise points are the outliers.

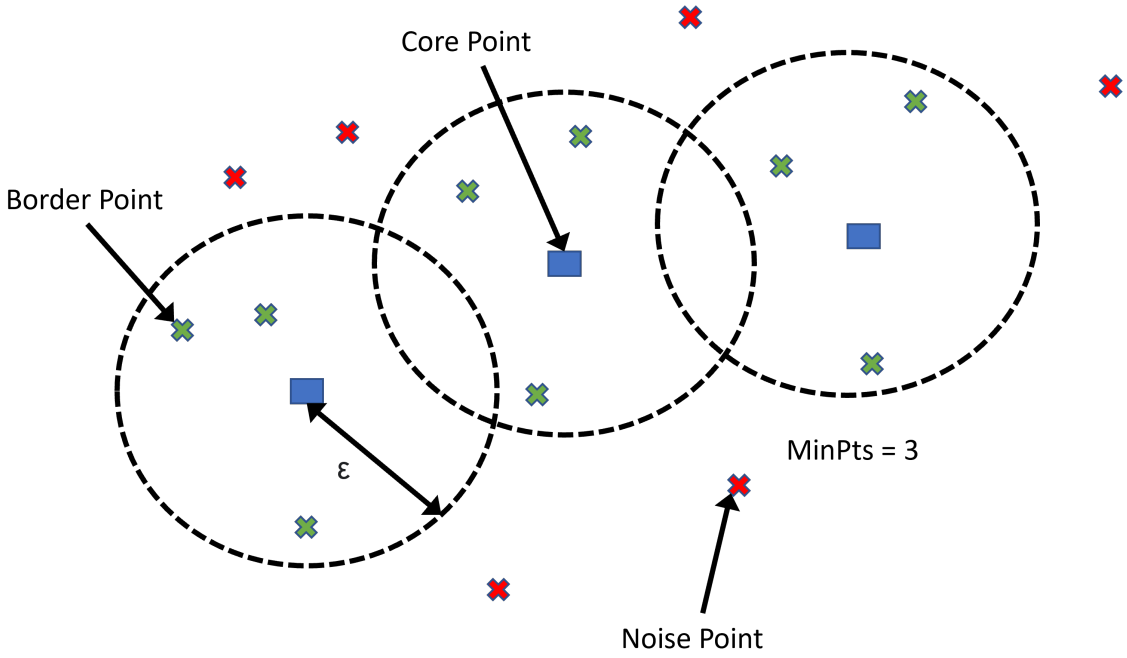


Figure 5.2: Chain of the DBSCAN algorithm. The two parameters are the ϵ and the minPts.

The DBSCAN algorithm is the following:

1. Identify the core points with more than minPts neighbors and the points in each point's *epsilon* neighborhood.

2. Find the connected parts of the neighbor graph's core points while disregarding all other nodes.

3. If the cluster has an *epsilon* radius, allocate each non-core point to it; if not, assign it to noise.

The goal of this module is to eliminate the saccades which caused by the user looking elsewhere but not at the object of interest. Figure 5.3 shows a visual example of the DBSCAN results. The green dots are the outliers, for example the user get distracted and looks elsewhere from the object, and the blue dots are the good gaze points on the object.

Figure 5.3: Bowl Place 5 Subject 2 DBSCAN parameters: $\epsilon = 0.01$, $min\_samples = 3$, good gaze points colour blue, outlier gaze points: green.

5.3.2 Kernel Density Estimation

The list of the aligned gaze fixations, $\hat{g}_{N,n}, n = N - \Delta t, .., N$, is the input of the Kernel Density Estimation (KDE) with Gaussian kernel (Pedregosa et al., 2011), which predicts the most probable location of the gaze fixation in the current frame. The KDE estimates the values as described in the following equation:

$$\rho_K(y) = \sum_{i=1}^{L_N} K(y - \hat{g}_{N,n,i}; h)$$

where a kernel, $K(x, h)$, is a positive function that is controlled by the bandwidth parameter, h. In our case, the bandwidth, h, parameter of the Gaussian kernel was set to 1, as default. L_N is the number of gaze points projected in the current frame N. The maximum of the estimated density surface is considered as a predictor of the gaze fixation point in the current frame. The search for the maximum is realized

inside a bounding box which encompasses all projected gaze fixations $\hat{g}_{N,n}, n = N - \Delta t, .., N$, using full search method with pixel accuracy. An example of an estimated gaze point in a frame is presented in Figure 5.4, depicted as the bright disk of the largest diameter.

Figure 5.4: Example of bowl place 4 subject 2 KDE gaze point estimation. The points are the gaze points and the white point is the estimated gaze point.

5.4 Depth estimation

For servoing the prosthesis arm towards the object-to-grasp, we need to compute the depth of the estimated gaze point. Depth extraction from monocular video or introduction of a supplementary depth sensor (a supplementary stereo camera, RGB depth sensor, etc.) into the system requires further investigation both in bio-physics and vision algorithms. Hence, in our work we propose an initial estimation of the depth of the gaze point from Tobii eye-trackers (AB, 2016) measures.

The depth estimation process is the following.

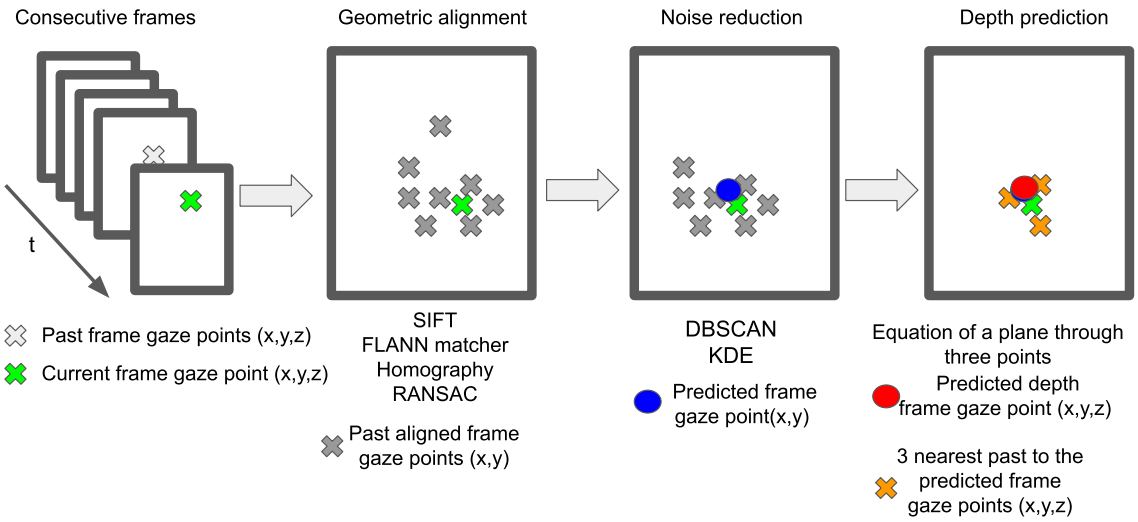


Figure 5.5: Depth estimation chain.

The Tobii eye-tracker recorded consecutive frames and gaze points, as Figure 5.5 shows. The recorded gaze points have 3 coordinates where x, y are the coordinates of location of the gaze points in a video frame and z is the distance between the recording unit and the object in millimeters. During the Geometric Alignment process (for more information see Chapter 5.2) the past aligned frame gaze points lost the depth information. So during the homography estimation just the location in the image coordinate system can be estimated, but not the depth.

The noise reduction step (for more information, see Chapter 5.3) has predicted frame gaze point location (x, y) in the current frame, just as depicted in Figure 5.5. For the gaze points originally recorded by eye-tracker system, the depth information z is known. Therefore, we propose to estimate the depth of the predicted gaze point in the current frame based on the equation of a plane through 3 points. The three points are the nearest gaze points in time to the predicted gaze points, with known depth information. The distance between these three points and the estimated gaze point in the current frame has to be small enough to ensure sufficient precision in

this solution, which uses a simple linear model. A plane is defined as:

$$ax + by + cz = d \tag{5.2}$$

where a, b, c, d are the coefficients. Let us consider the vectors V_1 and V_2:

$$\begin{aligned} V_1 &= P_3 - P_1 \\ V_2 &= P_2 - P_1 \end{aligned} \tag{5.3}$$

where P_1, P_2, P_3 are the 3 closest gaze points to the predicted gaze point. Then

$$\begin{aligned} V_1 &= v_{11}i + v_{12}j + v_{13}k \\ V_2 &= v_{21}i + v_{22}j + v_{23}k \\ M = V_1 \times V_2 &= \begin{vmatrix} i & j & k \\ v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{vmatrix} \\ &= (v_{12}v_{23} - v_{13}v_{22})i + (v_{13}v_{21} - v_{11}v_{23})j + (v_{11}v_{22} - v_{12}v_{21})k \\ a &= (v_{12}v_{23} - v_{13}v_{22}) \\ b &= (v_{13}v_{21} - v_{11}v_{23}) \\ c &= (v_{11}v_{22} - v_{12}v_{21}) \\ d = M \cdot P_3 &= \sum_{n=1}^{3} m_i p_i = \\ &= (v_{12}v_{23} - v_{13}v_{22})p_{31} + (v_{13}v_{21} - v_{11}v_{23})p_{32} + (v_{11}v_{22} - v_{12}v_{21})p_{33} \end{aligned} \tag{5.4}$$

The a, b, c coefficients are obtained from a vector normal to the plane, which is calculated as the cross-product of two vectors connecting the points. The d is from the dot product of the normal vector with any of the point position vectors.

The estimated gaze points depth is predicted based on the obtained plane equation.

$$z = \frac{ax + by - d}{-c} \tag{5.5}$$

where z is the depth and the a, b, c, d coefficients are calculated as in 5.4.

5.5 Results

In this section, results are presented. The first part explains the gaze point geometric alignment time measurements. The second part of this section is the gaze point noise reduction, which are the DBSCAN clustering and the KDE. In Chapter 5.5.3 the parameters were defined and justified for the DBSCAN clustering. It also contains our time measurements.

5.5.1 Geometric alignment measurements

The time measures of the geometric alignment module are given in Table 5.1. The OpenCV (Bradski, 2000) library 4.5.5 version was used during this experiment. The geometric alignment consists of an SIFT (Lowe, 2004) keypoint extractor, a FLANN matcher (Muja et Lowe, 2009), and a homography estimator. In the first part of Table 5.1, we give measures on embedded mobile ZCUs. The left-most column of Table 5.1 contains the name of the video file. The SIFT points have been detected in the mask, centred on the estimated gaze fixation point in each frame. The radius of the mask was chosen to encompass approximately 100 points. The second column contains the mean mask radius with standard deviation. For the geometric alignment by homography, we detected keypoints in two video frames: the current and previous reference frames. In the next columns, we give time figures on ARM A53 processors for keypoint (KP) computation on one frame, the matching

time, and homography computation time.

In Table 5.1, the second column contains the number of detected SIFT points with the corresponding mask radius. We also present it as the mean and standard deviation on the whole video. The time figures are given for general purpose Intel processors.

The matcher, the homography estimator, and the gaze projection on ZCU102 are fast enough for real-time processing, as illustrated in Table 5.1. The worst-case scenario was 0.024 s, for the FLANN matcher (Muja et Lowe, 2009), which means that the frame rate does not exceed 40 fps. This speed is enough for controlling a robotic arm.

However, the SIFT keypoint extractor was slower than the required processing time. While the worst-case scenario on the Intel i5 7300HQ CPU took 0.072 s, which is around 13.81 fps, on the ARM A53, it took 0.866 s, which is around 1.15 fps. For real-time processing, a rate of least 10 fps is required.

Table 5.1: Comparison between the Intel i5 7300HQ and the Xilinx ZCU102 ARM CORTEX A53 in the whole gaze alignment chain.

		Xilinx ZCU102 ARM CORTEX A53			
Video File Name	**Mask Radius**	**SIFT KP Extractions (ms)**	**Matcher (ms)**	**homography (ms)**	**Gaze Projection (ms)**
BowlPlace1Subject1	119 ± 25	875.504 ± 12.123	23.471 ± 5.203	2.200 ± 0.540	0.089 ± 0.004
BowlPlace1Subject2	106 ± 16	875.282 ± 9.504	20.036 ± 3.704	1.900 ± 0.398	0.088 ± 0.001
BowlPlace1Subject3	153 ± 50	873.072 ± 7.283	17.626 ± 3.276	2.539 ± 0.621	0.089 ± 0.001
BowlPlace1Subject4	120 ± 25	873.545 ± 9.062	22.244 ± 5.938	2.160 ± 0.464	0.092 ± 0.009
BowlPlace4Subject1	158 ± 55	855.947 ± 6.583	16.011 ± 3.053	2.883 ± 1.188	0.088 ± 0.001
BowlPlace4Subject2	117 ± 24	861.933 ± 5.821	16.276 ± 2.623	1.997 ± 0.449	0.089 ± 0.004
BowlPlace4Subject3	108 ± 19	867.649 ± 8.894	15.679 ± 4.620	2.136 ± 0.350	0.089 ± 0.005
BowlPlace4Subject4	147 ± 49	857.271 ± 9.468	16.762 ± 4.186	2.240 ± 0.516	0.088 ± 0.001
BowlPlace5Subject1	120 ± 33	861.481 ± 8.012	17.875 ± 2.176	2.018 ± 0.505	0.088 ± 0.001
BowlPlace5Subject2	133 ± 42	858.547 ± 6.232	17.944 ± 3.024	2.354 ± 0.880	0.088 ± 0.001
BowlPlace5Subject3	126 ± 33	859.774 ± 6.384	15.742 ± 2.836	2.007 ± 0.524	0.087 ± 0.001
BowlPlace6Subject1	120 ± 25	867.344 ± 10.950	19.026 ± 3.862	1.965 ± 0.306	0.088 ± 0.001
BowlPlace6Subject2	129 ± 35	862.750 ± 9.731	19.737 ± 4.973	3.681 ± 3.456	0.090 ± 0.008
BowlPlace6Subject3	127 ± 31	864.429 ± 6.931	17.555 ± 3.806	2.588 ± 0.823	0.087 ± 0.001
BowlPlace6Subject4	112 ± 22	867.962 ± 9.579	17.368 ± 4.725	2.710 ± 0.649	0.089 ± 0.004
			Intel i5 7300HQ		
Video File Name	**Number of Extracted KP**	**SIFT KP Extractions (ms)**	**Matcher (ms)**	**homography (ms)**	**Gaze Projection (ms)**
BowlPlace1Subject1	151 ± 67	74.205 ± 5.611	3.891 ± 0.853	0.259 ± 0.051	0.015 ± 10^{-4}
BowlPlace1Subject2	156 ± 37	75.062 ± 5.640	3.304 ± 0.579	0.228 ± 0.040	0.014 ± 10^{-4}
BowlPlace1Subject3	86 ± 50	72.217 ± 2.572	3.011 ± 0.476	0.282 ± 0.055	0.014 ± 10^{-4}
BowlPlace1Subject4	138 ± 69	72.979 ± 2.853	3.717 ± 0.940	0.252 ± 0.044	0.015 ± 0.002
BowlPlace4Subject1	94 ± 50	70.068 ± 2.405	2.747 ± 0.565	0.313 ± 0.113	0.014 ± 10^{-4}
BowlPlace4Subject2	121 ± 28	72.280 ± 3.538	2.778 ± 0.407	0.233 ± 0.040	0.015 ± 10^{-4}
BowlPlace4Subject3	126 ± 39	73.402 ± 3.406	2.678 ± 0.728	0.256 ± 0.047	0.014 ± 10^{-4}
BowlPlace4Subject4	95 ± 50	70.394 ± 2.349	2.872 ± 0.695	0.259 ± 0.051	0.014 ± 10^{-4}
BowlPlace5Subject1	129 ± 39	71.990 ± 2.691	3.027 ± 0.369	0.244 ± 0.050	0.015 ± 10^{-4}
BowlPlace5Subject2	120 ± 56	71.587 ± 2.526	3.077 ± 0.573	0.272 ± 0.087	0.014 ± 10^{-4}
BowlPlace5Subject3	108 ± 36	71.359 ± 2.500	2.684 ± 0.448	0.234 ± 0.049	0.015 ± 0.001
BowlPlace6Subject1	132 ± 48	72.150 ± 2.891	3.213 ± 0.645	0.237 ± 0.031	0.015 ± 10^{-4}
BowlPlace6Subject2	129 ± 59	71.790 ± 3.934	3.348 ± 0.823	0.390 ± 0.316	0.015 ± 10^{-4}
BowlPlace6Subject3	114 ± 47	72.042 ± 2.883	2.976 ± 0.617	0.287 ± 0.076	0.015 ± 0.001
BowlPlace6Subject4	138 ± 44	74.585 ± 4.431	3.089 ± 0.849	0.303 ± 0.075	0.015 ± 10^{-4}

5.5.2 Kernel Density Estimation without clustering and gaze points reduction window

Table 5.2 illustrates a comparison of the estimated time of KDE computation between the Intel i5-7300HQ and the Xilinx ZCU102 ARM Cortex A53. The second column contains the available number of gaze points during a frame gaze point estimation. The Intel i5-7300HQ (Intel, 2017) computes the KDE at 80 fps on average, and the ARM A53 (Xilinx, 2021a) computes the KDE at 7.9 fps on average. In some critical cases, when the scattering of the subject's gaze fixations is too strong, then the computation time is higher than in real-time, and is 3.9 s per frame, see the "Lid" sequence. Evidently, in such a case of highly cluttered scenes and problems of ocular movements, our system shows its limits.

Table 5.2: Comparison in processing time of kernel density estimation module between the Intel i5 7300HQ and the Xilinx ZCU102 ARM CORTEX A53.

		Xilinx ZCU102 ARM CORTEX A53		**Intel i5 7300HQ**	
Video File Name	**Gaze Points**	**Time (ms)**	**Max Time (ms)**	**Time (ms)**	**Max Time (ms)**
Bowl	22 ± 8	49.27 ± 82.83	307.34	4.94 ± 7.68	27.90
CanOfCocaCola	26 ± 11	75.54 ± 95.89	395.08	7.46 ± 8.80	36.70
FryingPan	24 ± 9	59.09 ± 50.06	206.76	5.86 ± 4.51	18.98
Glass	29 ± 10	148.22 ± 265.60	943.19	14.89 ± 26.23	92.21
Jam	27 ± 12	132.75 ± 319.01	1365.65	13.34 ± 31.39	134.68
Lid	29 ± 16	247.21 ± 718.32	3835.30	23.92 ± 70.97	379.64
MilkBottle	28 ± 10	114.95 ± 148.60	647.86	11.20 ± 13.99	61.92
Mug	28 ± 11	109.88 ± 218.40	1087.39	11.03 ± 21.26	106.63
OilBottle	30 ± 12	235.15 ± 477.79	2117.26	22.86 ± 46.23	205.83
Plate	32 ± 14	203.39 ± 406.91	1837.70	19.59 ± 39.46	178.97
Rice	29 ± 13	90.34 ± 95.16	372.93	8.64 ± 8.92	35.80
SaucePan	25 ± 12	139.07 ± 261.08	1286.11	13.68 ± 25.82	126.92
Sponge	24 ± 10	50.05 ± 49.79	207.89	5.10 ± 4.76	20.46
Sugar	27 ± 14	146.60 ± 271.58	1165.44	14.46 ± 26.70	117.57
VinegarBottle	28 ± 13	122.32 ± 178.37	683.56	12.23 ± 17.71	70.01
WashLiquid	28 ± 12	102.93 ± 183.02	880.47	10.42 ± 18.45	89.25

The problem is caused by outlier gaze fixation points, which fall far away from

the majority, increasing the KDE search area. The solution might be to use a simple clustering algorithm to find the outlier gaze fixation points and discard them. Practically, we use only the last, 10 gaze fixation points. Thus, we believe that this clustering can be carried out in a short time.

However, if the projected gaze fixations in the current frame are sufficiently close (in the radius of 10 pixels approximately accordingly to our observations, which is the “normal case"), the ARM A53 (Xilinx, 2021a) can compute the KDE in real time.

5.5.3 Kernel Density Estimation with DBSCAN clustering and Gaze Points reduction window

First, it has to estimate the maximum area of the KDE. Table 5.3 shows the results of the experiment of a reduced number of gaze points (10) on a Xilinx ZCU 102 embedded ARM CPU. When the maximum area set to 400 px the computational time was 7.68 ms. If the area is bigger, then more computational time is needed. This experiment proved that the 4900 px area size hits our target for real-time processing, because the DBSCAN clustering algorithm runs in 28.88 ms. This is 34.72 fps which is enough computational speed for us.

Table 5.3: Average computational time of the KDE after 100 iterations on the Xilinx ZCU 102 Embedded ARM CPU. The gaze points number is 10 and the size of the area is given.

10 Gaze Points measured on arm CPU after 100 iterations						
Area (px)	400	900	1600	2500	3600	4900
avg time (ms)	7.68	10.14	13.4	18.3	22.96	28.88

After the maximum area size is estimated, the next step is to find the best parameters for min_samples and ϵ for the DBSCAN clustering. They should be such that the area for KDE occupy a rectangle area of 4900 pixels or fewer.

Table 5.4: DBSCAN without setting the gaze point maximum number to 10.

ϵ	**min_samples**	**good videos**
0.005	3	0.79
0.005	4	0.75
0.005	5	0.71
0.01	**3**	**0.87**
0.01	4	0.86
0.01	5	0.85
0.05	3	0.54
0.05	4	0.54
0.05	5	0.54
0.1	3	0.51
0.1	4	0.51
0.1	5	0.51

The Table 5.4 shows our experiments when all the gaze points are used in a given GITW (LaBRI, 2016) video sequence. When the DBSCAN clustering is used with ϵ=0.01 and min_sample=3 parameters, 87% of the gaze points are clustered in a rectangular area smaller than 4900 px on the GITW dataset.

Table 5.5: DBSCAN with gaze point maximum number set to 10.

ϵ	**min_sample**	**good videos**
0.005	3	0.91
0.005	4	0.9
0.005	5	0.9
0.01	**3**	**0.95**
0.01	4	0.94
0.01	5	0.92
0.05	3	0.94
0.05	4	0.95
0.05	**5**	**0.96**
0.1	3	0.89
0.1	4	0.89
0.1	5	0.89

When the number of gaze points was fixed to maximum 10, the Table 5.5 shows the results. The best result is when the DBSCAN parameters are ϵ=0.05 and

min_sample = 5. In that case, the DBSCAN clusters 96% of the gaze points in a 4900 px area or less window on the GITW dataset. The ϵ=0.01, min_sample = 3 and the ϵ=0.05, min_sample=4 are also giving a promising result, in that case the 95% of the gaze points are clustered in a 4900 px area or less window on the GITW dataset.

In our case, the number of gaze points (GPs) was fixed to maximum 10, so the best parameters for the DBSCAN are the ϵ=0.05 and min_sample = 5.

Table 5.6: KDE + DBSCAN with removed outliers (Gaze Points area size <4900 px), DBSCAN parameters: ϵ=0.05, *min_samples*=5.

Name	**Number of GP before the KDE + DBSCAN**	**Number of GP after the KDE + DBSCAN**	**Computational time (ms)**	**Max computational time (ms)**	**Area of the BBs**	**Max Area**
Bowl	10.000 ± 0.000	8.765 ± 1.888	8.499 ± 3.520	18.614	614.6 ± 1032.6	3456
CanOfCocaCola	10.000 ± 0.000	7.500 ± 2.007	6.828 ± 0.929	9.773	206.1 ± 237.4	990
FryingPan	9.920 ± 0.277	7.320 ± 2.249	7.468 ± 3.127	18.154	384.2 ± 844.2	3337
Glass	9.962 ± 0.196	7.615 ± 2.155	6.725 ± 0.931	10.442	197.7 ± 255.9	1248
Jam	9.882 ± 0.485	8.529 ± 1.940	7.166 ± 2.242	15.672	312.5 ± 638.1	2745
Lid	9.964 ± 0.189	8.036 ± 2.219	8.225 ± 4.261	22.438	620.3 ± 1256.4	4753
MilkBottle	10.000 ± 0.000	7.955 ± 1.558	6.763 ± 0.681	8.640	207.8 ± 183.9	744
Mug	10.000 ± 0.000	8.214 ± 2.007	7.127 ± 2.126	17.385	314.1 ± 617.3	3312
OilBottle	10.000 ± 0.000	8.154 ± 2.034	7.104 ± 1.592	14.249	304.1 ± 470.6	2460
Plate	10.000 ± 0.000	7.667 ± 2.148	7.519 ± 2.538	16.518	435.1 ± 768.5	3344
Rice	9.808 ± 0.981	7.962 ± 2.163	7.508 ± 3.252	21.239	428.4 ± 969.0	4440
SaucePan	9.778 ± 0.847	7.148 ± 1.975	6.887 ± 0.965	9.683	254.0 ± 252.6	990
Sponge	10.000 ± 0.000	7.960 ± 2.111	6.817 ± 0.701	8.332	225.5 ± 177.4	624
Sugar	9.913 ± 0.417	7.826 ± 1.922	8.716 ± 4.707	23.154	752.2 ± 1340.0	4736
VinegarBottle	10.000 ± 0.000	7.321 ± 2.195	7.202 ± 1.468	11.697	346.8 ± 442.5	1840
WashLiquid	9.955 ± 0.213	8.545 ± 1.993	6.888 ± 0.973	10.202	254.0 ± 274.1	1224

Table 5.6 shows the computational time of the KDE + DBSCAN algorithm when the number of gaze points is fixed to 10, the DBSCAN was used with $\epsilon = 0.05$ and min_sample=5 and the maximum area of the gaze points after the DBSCAN is 4900px. The results show that in every subdataset the maximum time is low enough for real-time processing. Thus, the Computational time (ms) column indicates that the variance is low, that is the time measurements are stable through video sequences and therefore the proposed parameter values are acceptable in a variety of situations for arm prosthesis control.

5.5.4 Bounding Box generation time measurements

The bounding box generation is fast on the Intel i5 7300HQ CPU. On average, 1 frame is processed in 0.42351 $\pm$ 0.01991 milliseconds, which is more than 2500 fps. The embedded ARM A53 processor is also fast enough to generate bounding boxes in real time. The average computation time was 2.659 $\pm$ 0.027 milliseconds, which is more than 376 fps.

5.6 Conclusion

Hence, in this chapter, we have described the pre-processing steps of the whole process of object recognition and localization for prosthesis control. Furthermore, we have proposed a simple solution for depth estimation of the predicted gaze point on the object, which is necessary for servoing of the prosthetic arm towards the target object to grasp.

We have measured computational time and optimized the pre-processing steps. The geometric alignment time measurements show that the bottleneck of this step is the SIFT (Lowe, 2004) keypoint extraction. Our FPGA SIFT implementation, see Chapter 4, allows to drastically reduce the computation time. The other parts of the gaze geometric alignment such as FLANN matching, homography estimation with RANSAC and gaze projection are fast enough for the Xilinx ZCU 102 embedded ARM CPU.

The gaze point noise reduction algorithm such as KDE and DBSCAN can also compute in real-time. Our experiments show that the best parameters for the GITW dataset are the $\epsilon = 0.05$ and min_sample = 5. In that case, the DBSCAN can cluster 96% of the gaze points on all videos in our rich in-the-wild data set correctly. We remind that for computation time constraints we consider the gaze points well

clustered in a video sequence if the maximum area of the rectangle covering the cluster remains less than 4900px. If the DBSCAN clustering is used, then the KDE computation cost is low and make it possible for creating a real-time application.

The bounding box generation around the gaze point is fast enough for the embedded Xilinx ZCU 102 ARM CPU. It can generate the bounding boxes with 376 fps.

The measurements show that the gaze point alignment steps are fast enough on the ARM Cortex A53 (Xilinx, 2021a)-embedded CPU, except the SIFT (Lowe, 2004) point extraction step. Therefore, the SIFT (Lowe, 2004) detection module is implemented on the programmable logic part of the Xilinx ZCU102 (Xilinx, 2019) FPGA board. (Fejér et al., 2022)

In the next Chapter 6 we will focus on the Gaze-Driven CNN (González-Díaz et al., 2019) algorithms and based on our measurements we propose a hybrid hardware-software solution for the acceleration.

Chapter 6

Hybrid solutions for object recognition with Gaze-Driven CNN

6.1 Introduction

Finding an object in an image is a computationally complex process. There are several methods to do that such as YOLO (Redmon et al., 2016), SSD (Liu et al., 2016), and Faster R-CNN (Ren et al., 2017). For more information about those object recognition methods, see subchapter 2.3. We have chosen the Gaze Driven CNN (González-Díaz et al., 2019), which is based on the Faster R-CNN (Ren et al., 2017).

González-Díaz et al. (2019) has proposed a system, which can recognize the object of interest in an image based on the user gaze. The Gaze-Driven CNN however is not running fast enough for real-time computing in an embedded CPU.

In this Chapter 6 we discuss an optimized Gaze-Driven CNN implementation we propose to achieve a real-time processing. We have decided our optimization steps based on our measurements, you can see them in Chapter 3.5.1 and our new measurements on an embedded CPU which you can find in section 6.3.4. Based on those experiments, we have decided which part of the network has to run on the

FPGA, and which one can run on the embedded CPU.

González-Díaz et al. (2019) system contains three different modules: the ResNet50, the Faster R-CNN and the MIL Aggregation. We have proposed a new layer to make the original architecture from González-Díaz et al. (2019) lighter, the Reduction Layer. It will be presented in section 6.2. We compared our solution with other object recognition methods in section 6.3.3

6.2 Gaze-Driven CNN implementation

This module recognizes the object type (e.g., bowl, pan, etc.) and localizes it in a first-person video frame. A limited number of bounding boxes of different scales is generated around the estimated gaze fixation point to localize an object. The module's input is thus the estimated gaze fixation point $\hat{g}_n$, the cropped frame around the estimated gaze fixation, and the possible bounding boxes which are generated around the estimated gaze point on the object $\hat{g}_n$-(–see the second block in Figure 3.2).

Accordingly, to González-Díaz et al. (2019) nine bounding boxes (BBs) have been generated with different scale and shape factors. The size of a cropped frame is 300 px $\times$ 300 px (González-Díaz et al., 2019). For the selection of the size of BB, we have considered the typical object size in accordance with the frame resolution, types of objects the person manipulates in the kitchen environment and the geometry of our visual scenes. Thus, the size in pixels of bounding boxes with different scales and shapes is chosen between 67 px and 223 px.

Recognition of the object is carried out by a CNN classifier applied to each of the generated bounding boxes. The BB with the maximum score is considered as object location.

Figure 6.1 shows the structure of the gaze-driven CNN. The backbone is a ResNet50 in the first four layers, see the block on the left in Figure 6.1. These layers serve as feature extractors from the input image. The input of the backbone is a cropped video frame of size 300 px × 300 px × 3. The output is a 1024 × 19 × 19 feature tensor. For more information about the ResNet50 see Chapter 6.2.1.

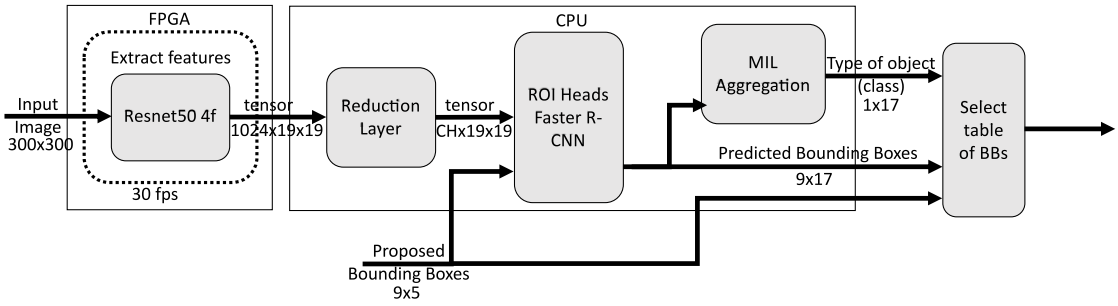


Figure 6.1: Gaze-driven, object-recognition CNN, where CH is the number of output channels of the Reduction Layer.

Not all feature channels are equally important for object classification when using the backbone. To select the most important ones, and to reduce the computational burden of the remaining part of the network, we introduce a Reduction Layer (RL). It reduces the number of channels in the input tensor to a given channel number CH (in our case, CH can be: 32, 64, 96, 128, 256, 512, 1024). For more information about the RL see in section 6.2.2.

Bounding boxes generated around the estimated gaze fixation point, and feature tensor with the reduced number of channels ($CH\times$ 19 × 19) are the inputs of the Faster R-CNN module (Girshick, 2015) (ROI Heads). The module predicts the object type and location as a 17 × 9 tensor as we have 9 BBs (see Figure 6.2 and work with a 17-class taxonomy comprising 16 object classes and a rejection class, as in González-Díaz et al. (2019). We present this par of our paper line in more details in section 6.2.3.

The class scores of bounding boxes are aggregated, as in González-Díaz et al.

(2019), by Multiple Instance Learning (MIL) (Amores, 2013). The module predicts the class of the object.

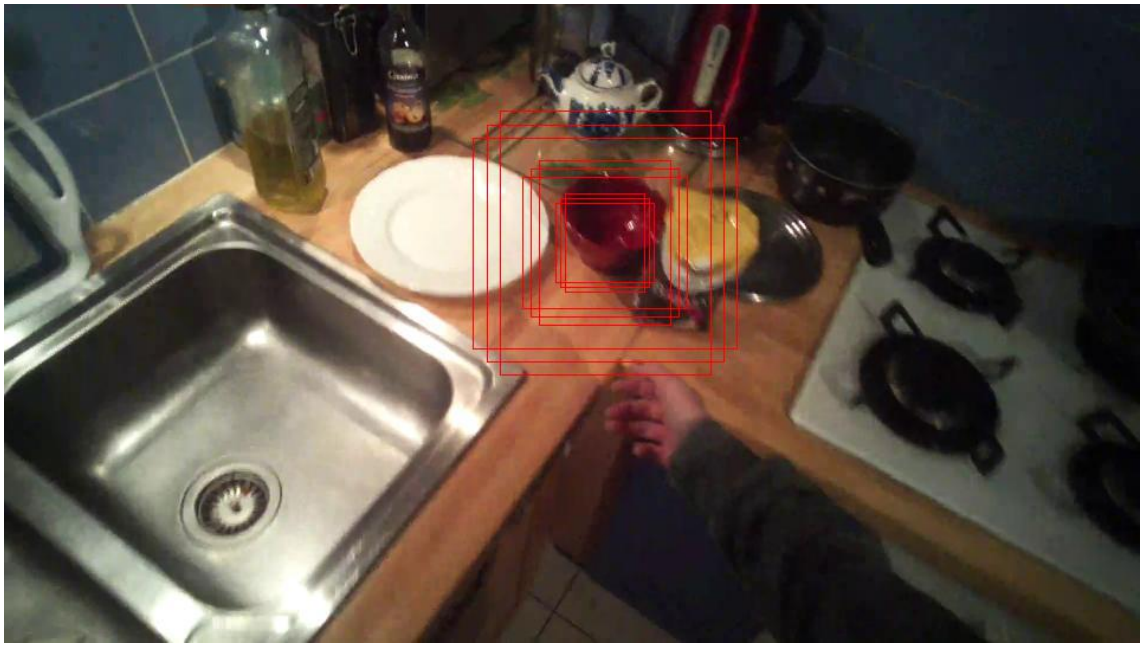


Figure 6.2: Example of "Bowl place 1, subject 1", (GITW) generated bounding box. The bounding boxes are generated around the red bowl.

6.2.1 Resnet50 implementation

The ResNet50 (He et al., 2016) is the backbone of the Faster R-CNN module of our implementation. The goal of this unit is to create a representative feature map from the object of interest.

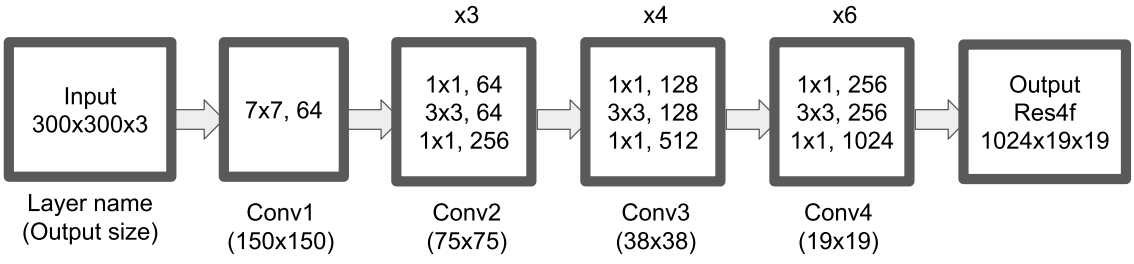


Figure 6.3: ResNet50 res4f architecture, shown with the residual units, the size of the filters and the outputs of each convolutional layer. Downsampling is performed by conv2_1, conv3_1, and conv4_1 with a stride of 2.

Figure 6.3 illustrates the architecture of the ResNet50 (He et al., 2016). The layers after res4f are discarded, and the network is initially established with its original weights up to that layer. The input of the backbone is a cropped video frame of size 300 px × 300 px × 3. The center of this cropped video frame is the estimated gaze point.

The residual blocks are also shown in Figure 6.3. The output is a 1024 × 19 × 19 feature tensor.

6.2.2 Reduction layer

When using the ResNet50 backbone, not all feature channels are equally crucial for object classification. We thus developed the RL to prioritize the most crucial ones and lessen the computational cost on the remaining portion of the network. It reduces the number of channels in a given input tensor to a target channel number CH. We have experimented with the following channel numbers CH: 32, 64, 96, 128, 256, 512, 1024.

The input of RL is the backbone output tensor of dimension 1024 × 19 × 19. The RL applies a 2D convolution (Paszke et al., 2019) over the input tensor. Assume that the input is of dimension (C_{in}, H, W) and the output is (C_{out}, H_{out}, W_{out}), then the RL can be precisely described as follows:

$$out(C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(k) \tag{6.1}$$

where $\star$ is the 2D cross-correlation operator, C denotes the number of channels, H is the height of input planes in pixels, and W is the width in pixels.

6.2.3 Faster R-CNN

The Faster R-CNN module's (Girshick, 2015) (ROI Heads) inputs are bounding boxes constructed around the estimated gaze fixation point and a feature tensor with reduced channels ($CH \times 19 \times 19$) by the RL. The module uses a 17-class taxonomy made up of 16 object classes and a rejection class, as in González-Díaz et al. (2019), to estimate the item type and position as a 17×9 tensor because there are 9 BBs. This tensor contains the probability of each bounding box for each class.

$$output_{ROIheads} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & ... & P_{1B} \\ P_{21} & P_{22} & P_{23} & ... & P_{2B} \\ & & ... & & \\ P_{C1} & P_{C2} & P_{C3} & ... & P_{CB} \end{bmatrix} \tag{6.2}$$

Equation (6.2) is the output tensor of the ROI heads (Faster R-CNN (Girshick, 2015)), where Ci are the categories and B are the bounding boxes.

In the network for "object proposals" or the ROIs the Multi-scale ROI Align pooling (Lin et al., 2016) is used on the Feautre Pyramid (FP). They assign ROIs of different scales to the FP levels, number by k. To compute the level index k the equation is the following:

$$k = \lfloor k_0 + log_2(\frac{\sqrt{wh}}{s}) \rfloor \tag{6.3}$$

Here k_0 is the target level on which a ROI with the size $wxh = 224^2$ should be mapped to. It is called "canonical level". In our experiments, the k_0 (canonical_level) was set to 4, and the s (canonical scale) was set to 224, which is the same as in Lin et al. (2016).

In the FPN, a 3 x3 convolution filter is applied over the feature maps for each scale level, then a separate 1x1 convolution is done for objectness predictions and

border box regression. The Region Proposal Network head refers to these 3x3 and 1x1 convolutional layers. All Pyr_k scale levels of feature maps are applied with the same head.

6.2.4 Multiple-Instance Learning implementation

Class scores with bounding boxes (equation 6.2) are aggregated by Multiple Instance Learning (MIL), as in González-Díaz et al. (2019) (Amores, 2013). The input of the MIL aggregation is the output tensor of the Faster R-CNN (Girshick, 2015). The module predicts the class of the frame, i.e. of the object. The frame-level score ($\hat{y}(f, c)$) is calculated as shown in Equation (6.4).

$$\hat{y}(f, c) = \frac{1}{\gamma} log(\sum_{b=1}^{BB_f} e^{\gamma y(b,c)}) \tag{6.4}$$

Here, f is the frame, c is the class, b is the bounding box, and $y(b, c)$ is the score of the bounding box. γ is a normalization factor.

MIL aggregation will produce the vector of the frame-level scores for the object categories. This vector can be finally transformed into the vector of object probabilities using a simple softmax operator: p(f, c) = softmax($\hat{y}$(f, c)).

6.3 Results

In this section, we discuss the measured computing time of the different steps of the proposed algorithm.

6.3.1 Dataset

The GITW (LaBRI, 2016) dataset contains egocentric videos recorded by a camera on the eye tracker glasses. It includes the gaze points of where the person was looking at each moment. The videos were recorded in the wild, in real kitchens, by different subjects, and every video was recorded by a subject who grasped a kitchen object.

The acquisition device used was Tobii Glasses 2 (eye tracker) with an egocentric scene camera. The Tobii Glasses video resolution is HD (1280 pixels $\times$ 720 pixels), and the video frame rate is 25 fps. There are 16 different kitchen objects in the videos: bowl, plate, wash liquid, vinegar bottle, milk bottle, oil bottle, glass, lid, saucepan, frying pan, and mug. Different subjects recorded the dataset in five different kitchens. The videos were short, around 10 s long. The GITW (LaBRI, 2016) dataset contains 404 videos overall. The dataset is freely available for research.

We carried out the time measurements on a subset of the GITW dataset, containing fifteen videos of "grasping a bowl" actions, recorded by four different subjects. The kitchen environments are of different complexity, from a scene with just a few objects, such as the BowlPlace1 videos, to a highly cluttered scene, such as BowlPlace4. The class bowl object had a strong inner variance: different colours, the material of the bowl object, and even a transparent one. The lighting conditions and the visibility are different. Moreover, sometimes, we obtained strong blurring effects due to the camera motion, which was worn on the person's body.

6.3.2 Gaze-Driven Object-Recognition CNN time measurements

Here, all measurements were taken by PyTorch. 1.6. (Paszke et al., 2019).

The measurements in Table 6.1 show that the most time-consuming part of the CNN is the ResNet50 backbone. In every case, the backbone can process a frame in 0.09 s on Intel i5 7300 CPU, which is equal to 11 fps. On the ARM A53 processor, see Table 6.2, this time, presented in the second column, is even higher. It is about 1.8 s, thus giving 0.5 fps. This is below the required computational speed. Higher channel number causes larger computational complexity in the Reduction Layer and the region of interest (ROI) heads, as shown in Tables 6.1 and 6.2. Nevertheless, with a reasonable number of channels after the reduction, not exceeding 128, these blocks run in real-time, with 82 fps for channel reduction and 25 fps for ROI heads.

Table 6.1: Measurements of the gaze-driven, object-recognition CNN in the Intel i5 7300 CPU. The first column contains the reaming number of channels after the Reduction Layer. Each column shows the elapsed time during the computation in milliseconds.

Number of Channel	**Backbone (ms)**	**Reduction Layer (ms)**	**ROI Heads (ms)**	**Aggregation (ms)**
32	90.000 ± 0.250	0.336 ± 10^{-4}	1.107 ± 10^{-4}	0.137 ± 10^{-6}
64	97.307 ± 1.613	0.531 ± 0.002	2.262 ± 0.004	0.138 ± 10^{-6}
96	87.441 ± 0.508	0.557 ± 0.003	2.956 ± 0.003	0.241 ± 10^{-4}
128	89.952 ± 2.568	0.646 ± 0.001	3.356 ± 0.001	0.142 ± 10^{-6}
256	85.287 ± 0.375	0.908 ± 10^{-4}	6.592 ± 0.002	0.150 ± 10^{-5}
512	94.505 ± 2.100	2.485 ± 0.002	12.276 ± 0.002	0.159 ± 10^{-6}
1024	95.515 ± 7.285	3.204 ± 0.007	23.718 ± 0.010	0.164 ± 10^{-6}

The slowest part of the system was, thus, the backbone; therefore, it was implemented in FPGA. The accelerated ResNet50 CNN on ZCU102 can process an image in 0.02686 s, which is 37.23 fps. This is high enough for real-time processing.

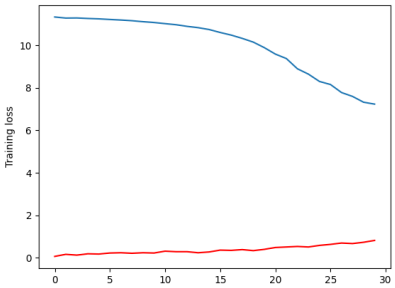
Table 6.2: Measurements of the gaze-driven, object-recognition CNN in the ARM A53 CPU. The first column contains the remaining number of the channels after the Reduction Layer. Each column shows the elapsed time during the computation in milliseconds.

Number of Channel	Backbone (ms)	Reduction Layer (ms)	ROI Heads (ms)	Aggregation (ms)
32	1863.300 ± 11.433	6.949 ± 0.001	13.843 ± 0.002	0.643 ± 0.001
64	1768.616 ± 15.615	8.156 ± 0.001	21.859 ± 0.006	0.708 ± 10^{-4}
96	1787.737 ± 15.903	10.178 ± 0.001	30.705 ± 0.001	0.758 ± 10^{-6}
128	1800.327 ± 17.915	12.140 ± 0.001	39.371 ± 0.002	0.727 ± 10^{-5}
256	1797.798 ± 16.372	22.061 ± 0.011	73.750 ± 0.002	0.714 ± 10^{-4}
512	1733.458 ± 14.429	33.723 ± 0.001	142.231 ± 0.001	0.752 ± 10^{-6}
1024	1761.748 ± 16.305	63.319 ± 0.001	285.121 ± 0.002	0.714 ± 10^{-6}

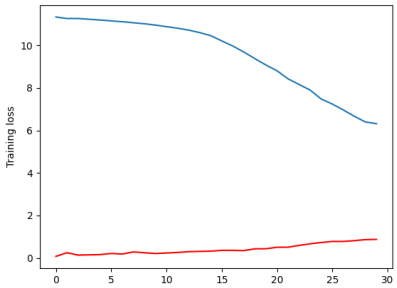
The measurements in Table 6.2 show the results of the ARM A53 CPU.

6.3.3 Gaze-Driven Faster R-CNN accuracy

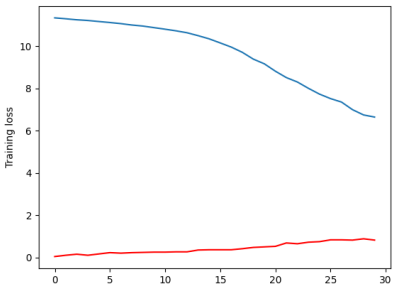
As Table 6.3 and Figure 6.4 show, the proposed architecture with reduction layer can perform sufficiently well on our real-world data. Reducing the number of channels to 128 does not impoverish the classification accuracy too much, compared with the initial 1024 feature channels of the backbone, as we can see from Table 6.3. The average accuracy and loss are computed per class of objects.



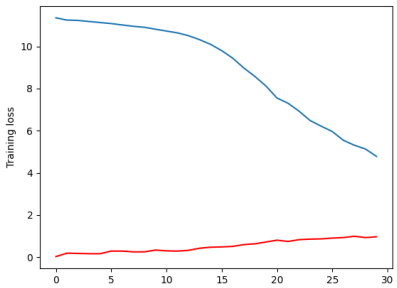
(**a**) Number of channels is 32



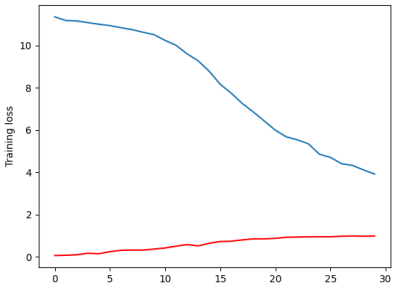
(**b**) Number of channels is 64



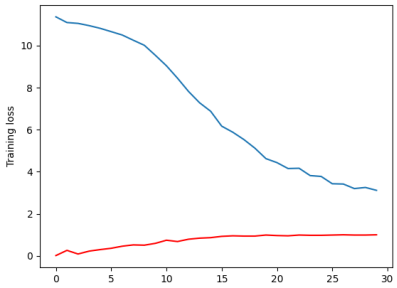
(**c**) Number of channels is 96



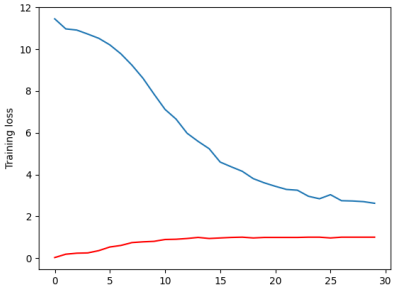
(**d**) Number of channels is 128



(**e**) Number of channels is 256



(**f**) Number of channels is 512



(**g**) Number of channels is 1024

Figure 6.4: Training accuracy (in red) and loss (in blue) during 30 epochs with different number of channels due to Reduction Layer.

Table 6.3: The results of the training and testing after 30 epochs.

Number of Channel	**32**	**64**	**96**	**128**	**256**	**512**	**1024**
avg loss on training set	7.235	6.318	6.642	4.778	3.920	3.115	2.623
avg acc on trainig set	0.815	0.877	0.827	0.963	0.988	1.000	1.000
avg acc on test set	0.793 ± 0.261	0.926 ± 0.120	0.853 ± 0.161	0.952 ± 0.083	1.000	1.000	1.000
avg ap on test set	0.978 ± 0.043	0.985 ± 0.030	0.964±0.041	0.995 ± 0.012	1.000	1.000	1.000

Table 6.4: Comparison of different object recognition CNNs. All the measurements were taken by Vitis AI 1.4. The gaze-driven, object-recognition CNN used 128 channels in the Reduction Layer.

Name	**Gaze-Driven, Object-Recognition CNN**	**SSD Mobilnet V2**	**YOLO V3**
Dataset	GITW	COCO	VOC
Framework	Pytorch	Tensorflow	Tensorflow
Input size	300 × 300	300 × 300	416 × 416
Running device	ZCU 102 + ARM A53	ZCU 102	ZCU 102
fps	12.64	78.8	13.2

Table 6.4 shows a comparison between different object recognition methods from the state-of-the-art methods and our method. The SOTA methods, such as lightweight YOLO V3 (Redmon et Farhadi, 2016) and SSD Mobilnet V2 (Liu et al., 2016), are trained on the COCO and VOC datasets. We have a specific and very cluttered kitchen environment. For this reason, we do not think that these object detectors are suitable in our case. From the computational time point of view (Xilinx, 2021c), implemented on the same architecture, they are a bit faster: 13.2 fps object recognition for YOLO V3 (Redmon et Farhadi, 2016) and 78.8 fps for SSD Mobilnet V2 (Liu et al., 2016). In our work, we take profit from the availability of gaze fixations in real-time, which can drive object localization. However, the actual implementation of KDE on CPU makes the system slower. We have 12.64 fps for object recognition and its localization. The bottleneck is the KDE estimation, which we are now improving. Nevertheless, our actual computation times are

compatible with real-time prosthesis control.

6.3.4 Time measurement of the whole system

Table 6.5 illustrates the average computational time of the system in milliseconds. The first column contains the module name, and the second column contains the Intel i5 7300HQ (Intel, 2017) CPU results. In the third column, the ARM A53 (Xilinx, 2021a)-embedded CPU results are given. The fourth column contains the hybrid (ZCU102 (Xilinx, 2019) and the ARM A53 (Xilinx, 2021a)) results.

Table 6.5: The average computational time measurement of the whole system on different hardware. The ResNet50 number of channels is 128.

	Computational Time (ms)		
Module name	**Intel i5 7300HQ CPU**	**ARM A53**	**FPGA + ARM A53**
SIFT (Lowe, 2004)	72.407 ± 3.349	865.499 ± 8.437	7.407 (Fejér et al., 2021a)
FLANN matcher	3.094 ± 0.638	18.223 ± 3.867	18.223 ± 3.867
homography estimation	0.270 ± 0.075	2.359 ± 0.778	2.359 ± 0.778
Gaze point projection	0.015 ± 10^{-4}	0.089 ± 0.003	0.089 ± 0.003
DBSCAN + KDE estimation	0.013 ± 0.003	7.34 ± 2.122	7.34 ± 2.122
bounding box generation	0.424 ± 0.020	2.659 ± 0.027	2.659 ± 0.027
ResNet50 (He et al., 2016)	89.952 ± 2.568	1800.327 ± 17.915	26.860
Reduction Layer	0.645 ± 0.001	12.140 ± 0.001	12.140 ± 0.001
Faster R-CNN (Ren et al., 2017)	3.356 ± 0.001	39.371 ± 0.002	39.371 ± 0.002
MIL Aggregation	0.142 ± 10^{-6}	0.727 ± 10^{-6}	0.727 ± 10^{-6}
Total time (ms)	170.298 ± 6.654	2748.734 ± 33.152	117.175 ± 6.8

The total computation time is 170.298 ms in the Intel i5 7300HQ, which is 5.872 fps. The ARM 53 (Xilinx, 2021a)-embedded CPU is the slowest because it is needed 2748.734 ms per frame, which is 0.364 fps. The hybrid embedded solution is computed in a frame of 117.175 ms, which is 8.534 fps. The hybrid embedded solution is faster than the Intel i5 7300HQ (Intel, 2017). The power consumption of the hybrid embedded solution is 5.6 W, which is less than the Intel i5 7300HQ (Intel, 2017) CPU 45 W.

The measurements show that the current experimental setup with the whole chain of modules is not yet suitable for real-time processing. However, with pipelining the modules, the real-time processing speed is achievable, but the latency will be higher.

6.4 Conclusion

In this Chapter 6, we have proposed a hybrid implementation of a visual analysis part for visual servoing of a prosthetic arm. The system was partitioned between the FPGA fabric and the ARM Cortex A53 processors of the Xilinx ZCU102 development board, based on the computing performance measurements of the building blocks. As a reference, the computing time of each image processing step was also measured on a laptop microprocessor and its power dissipation was estimated. (Fejér et al., 2022)

The gaze-driven CNN is built on 4 different modules: ResNet50 (He et al., 2016), Reduction Layer, Faster R-CNN (Ren et al., 2017), and Multiple Instance Learning (MIL) aggregation. ResNet50 (He et al., 2016) was accelerated on FPGA because the measured computational speed on the ARM Cortex A53 processor was only 0.55 fps, which was improved to 37.23 fps. The Faster R-CNN is also slow, providing only 3.5 fps when the number of input channels is 1024. We thus proposed a new Reduction Layer between the ResNet50 (He et al., 2016) and the Faster R-CNN (Ren et al., 2017) to reduce the number of input channels for the latter block. The frame rate can be increased to 25 fps when the number of input channels of the Reduction Layer in the Faster R-CNN is reduced to 128. The experiments show that the accuracy using only 128 channels is still high enough for the bounding box classification. (Fejér et al., 2022)

The experimental setup, with the whole chain of modules, is not suitable for real-time processing (117.175 ms on average, or approximately 8.5 fps). However, this computing time can be improved by pipelining the system and processing different frames at each stage, because each block can finish processing an image within 40 ms. The drawback of pipelining is increased latency.

The power consumption and processing speed for the different architectures show that the embedded system, accelerated with FPGA, is a feasible solution for creating a wearable device. (Fejér et al., 2022)

Chapter 7

Conclusion

In this chapter, I will present my main results and outline the perspectives of this work.

In this PhD research, I developed a full solution for object recognition with Deep NNs in ego-centered video on a hybrid architecture using FPGA. The solution is guided by eye-tracker gaze fixations recordings and is designed for visual servoing of upper limb neuroprostethic arms. In the following, I summarize my contributions.

7.1 New scientific results

My scientific results are two-fold:

- 1. I developed a hybrid solution: FPGA-CPU - for object recognition in ego-centered video by a Gaze-Driven CNN.

- 2. As a re-usable part of it I have implemented, on FPGA, a new SIFT detector for pre-processing of gaze data, namely their alignment in the current video frame.

I present my contributions in the following theses.

Thesis 1: A hybrid solution for Gaze-Driven CNN. (Fejér et al., 2022)

The backbone of the solution is ResNet50 (He et al., 2016). A Reduction Layer has been introduced to accelerate computations. Faster R-CNN (Ren et al., 2017), is implemented for classification of object proposals, i.e. candidates bounding boxes in the current frame fitting the object of interest. Multiple Instance Learning (MIL) aggregation is applied for fusion of individual classification results to get the overall object score and position in the current frame. The recognition of an object type and object localization in a current video frame is running almost in real-time.

Thesis 1.1 Based on a critical analysis of computation complexity hybrid implementation of building blocks are proposed. (Fejér et al., 2022)

I have conducted a detailed computational time analysis of software implementation of object detection and localization method proposed by LaBRI, González-Díaz et al. (2019). I have partitioned the system between the FPGA fabric and the ARM Cortex A53 processors of the Xilinx ZCU102 development board, based on the computing speed measurements of the building blocks. As a reference, the computing time of each image processing step was measured on a laptop microprocessor and its power dissipation was estimated.

The gaze point alignment is fast enough according to my measurements on the ARM Cortex A53 (Xilinx, 2021a)-embedded CPU, except the SIFT (Lowe, 2004) point extraction step. Therefore, I have implemented the SIFT detection module on the programmable logic part of the Xilinx ZCU102 (Xilinx, 2019) FPGA board

The gaze-driven CNN is built on four different modules: ResNet50 (He et al., 2016), Reduction Layer, Faster R-CNN (Ren et al., 2017), and Multiple Instance Learning (MIL) aggregation. I accelerated the ResNet50 (He et al., 2016) on FPGA with Vitis AI because the measured computational speed on the ARM Cortex A53 processor was only 0.55 fps. I have improved it to 37.23 fps due to this implementation.

Thesis 1.2: I proposed a new Reduction Layer between ResNet50 and Faster R-CNN in the original algorithm to reduce the number of input channels for the Faster R-CNN block. (Fejér et al., 2022)

The frame rate can be increased to 25 fps when the number of input channels for the Faster R-CNN is reduced to 128 by the Reduction Layer. The experiments show that the accuracy using only 128 channels is still high enough for the bounding box classification.

Thesis 1.3: My hybrid FPGA + ARM solution show that a wearable device is achievable with low power dissipation and with real-time processing speed. (Fejér et al., 2022)

My experimental setup, with the whole chain of modules, ensures the computational speed of 117.175 ms on average per video frame. That is, the computational frame rate is approximately 8.5 fps. It is yet not suitable for real-time processing, as the required video frame-rate for servoing of a prosthetic arm is 10 fps. However, this computing time can be improved by pipelining the system. Because each block of the system can finish processing of a video frame within 40 ms, according to our measurements. This means that we can achieve a frame rate of 25 fps. However, the latency of the system will be increased to 117.175 ms, which is still affordable in this scenario.

Thesis 2: I developed a hybrid solution with a 32 bit floating point computations for SIFT keypoint extractor. It runs on FPGA and generates the same results as the OpenSIFT software implementation. (Fejér et al., 2021a)

In the overall system SIFT (Lowe, 2004) point detection is used for the alignment of gaze points. It is a critical block according to our computation time measurements,

because the processing time of a frame was 72.407 ± 3.349 ms on Intel i5 7300HQ CPU. Therefore, we have designed, optimized and implemented it on FPGA.

Thesis 2.1: My proposed FPGA solution for the SIFT point detector can be implemented on Xilinx ZCU102 FPGA-board. (Fejér et al., 2021a)

I have implemented a FPGA-optimizied SIFT keypoint detector on Xilinx ZCU102. The designed digital circuit has been used 117,620 of the 274,080 available LUTs resources, 157,946 of the 548,160 available FFs resources, 416.5 of the 912 available BRAMs resources and 938 of the 2,520 available DSPs resources. There are enough free resources left on the Xilinx ZCU102 to develop more computer vision algorithms on FPGA.

Thesis 2.2: In the proposed FPGA SIFT implementation, I have made a simplification in the keypoint localization step. The FPGA optimzed SIFT which gives close results to the reference SIFT point detector. (Fejér et al., 2021a)

Instead of computing Taylor expansion for precise SIFT point localization, keypoints too close to each other are filtered using Non-maximum Supression. This approach is not changing the accuracy of the original algorithm accordingly to our comparison. I have compared my hardware/software solution to other hardware or hybrid implementations of the SIFT algorithm and with the baseline software detector OpenSIFT. I have conducted computational experiments on a large set of 3860 video frames to validate the implemented detector. My algorithm implemented on FPGA is giving an average precision of 0.84 and the average recall of 0.94 in SIFT-point detection compared to the baseline OpenSIFT.

Thesis 2.3: I have experimentally shown that the proposed FPGA SIFT implementation is time-efficient and the power consumption is low. The processing rate of the implemented SIFT detector is 135 images per

second, when the input image resolution is of 480px × 480px, and it is running on Xilinx ZCU102 FPGA board. The total power consumption of my FPGA SIFT implementation is 5.6W, which is suitable for wearable devices. (Fejér et al., 2021a)

I used these results in Thesis 1, because in one of the steps, namely in gaze point alignment the SIFT keypoint extraction is a crucial step.

7.2 Perspectives

The system integration is possible, because the obtained accuracy is high enough for real world demonstration. Currently, the visual block is accelerated on FPGA, but it is necessary to implement the system servoing steps on an embedded hybrid system.

The algorithm can also be developed further. The current algorithm is extracting objects in video frame by frame. However, it is possible to achieve higher accuracy with tracking. In that case, faster processing time is also achievable. Another possible future work will be to use move-to-data incremental learning method (Poursanidis et al., 2020). With this method, it is possible to adapt this system for a changing living environment of a person. It will also be adaptable for a different environment.

A wearable device is required in that case to control a prosthetic arm. To do that, a technology needed has to have a low power consumption and a high computational speed. It is feasible, as my experiments have shown.

It is possible to increase the computational speed with pipelining. Pipelining is a method when a step finishes, for example gaze-point alignment of a frame, then it is possible to start gaze-point alignment on the next frame and the gaze-point noise

reduction can be done on the original frame at the same time. However, this would increase the latency of the system. The current system latency is around 117.175 ± 6.8 ms, which is the accepted latency allowed by the control of the robotic arm (∼100 ms).

Bibliography

[AB 2016] AB, Tobii : *Tobii Pro Glasses 2 API - Developer's Guide*. 09 2016. – URL https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/. – accessed on: 2022-07-16 52, 55, 122

[Amores 2013] Amores, Jaume : Multiple instance classification: Review, taxonomy and comparative study. In : *Artificial Intelligence* 201 (2013), pp. 81–105. – URL https://www.sciencedirect.com/science/article/pii/S0004370213000581. – ISSN 0004-3702 27, 137, 140

[Bay et al. 2008] Bay, Herbert ; Ess, Andreas ; Tuytelaars, Tinne ; Van Gool, Luc : Speeded-Up Robust Features (SURF). In : *Comput. Vis. Image Underst.* 110 (2008), June, n. 3, pp. 346–359. – URL http://dx.doi.org/10.1016/j.cviu.2007.09.014. – ISSN 1077-3142 40, 48, 83

[Björkman et al. 2014] Björkman, Mårten ; Bergström, Niklas ; Kragic, Danica : Detecting, segmenting and tracking unknown objects using multi-label MRF inference. In : *Computer Vision and Image Understanding* 118 (2014), pp. 111 – 127. – URL http://www.sciencedirect.com/science/article/pii/S107731421300194X. – ISSN 1077-3142 40, 41, 108

[Blott et al. 2018] Blott, M. ; Preusser, T. B. ; Fraser, N. J. ; Gambardella, G. ; O'Brien, K. ; Umuroglu, Y. ; Leeser, M. ; Vissers, K. : FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. In : *ACM Trans. Reconfigurable Technol. Syst.* 11 (2018), December, n. 3 78

[Bradski 2000] Bradski, G. : The OpenCV Library. In : *Dr. Dobb's Journal of Software Tools* (2000) 40, 90, 92, 114, 115, 116, 125

[Buzási 2018] Buzási, Bence : *Depth estimation for robotic prosthesis arm with object-to-grasprecognition from eye-tracker glasses, application to Neuroprothesis control*, UBx, PPCU, UAM, Diploma thesis, 7 2018 26

[Calonder et al. 2010] Calonder, Michael ; Lepetit, Vincent ; Strecha, Christoph ; Fua, Pascal : BRIEF: Binary Robust Independent Elementary Features. In : *Proceedings of the 11th European Conference on Computer Vision: Part IV*. Berlin, Heidelberg : Springer-Verlag, 2010, pp. 778–792. – URL `http://dl.acm.org/citation.cfm?id=1888089.1888148`. – ISBN 3-642-15560-X, 978-3-642-15560-4 33, 40

[Chang et al. 2020] Chang, Eunhee ; Kim, Hyun T. ; Yoo, Byounghyun : Virtual Reality Sickness: A Review of Causes and Measurements. In : *International Journal of Human–Computer Interaction* 36 (2020), n. 17, pp. 1658–1682. – URL `https://doi.org/10.1080/10447318.2020.1778351` 33

[Chang et al. 2013] Chang, Leonardo ; Hernández-Palancar, José ; Sucar, L. E. ; Arias-Estrada, Miguel : FPGA-based detection of SIFT interest keypoints. In : *Machine Vision and Applications* 24 (2013), Feb, n. 2, pp. 371–392. – URL `https://doi.org/10.1007/s00138-012-0430-8`. – ISSN 1432-1769 40, 43, 89, 109

[Chollet 2017] Chollet, François : Xception: Deep Learning with Depthwise Separable Convolutions. In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1800–1807 45

[Chum et Matas 2005] Chum, O. ; Matas, J. : Matching with PROSAC - progressive sample consensus. In : *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* 1, 2005, pp. 220–226 vol. 1 117

[da Costa Barreiros 2020] Costa Barreiros, João C. da : *Fast Scale-Invariant Feature Transform on GPU*, Universidade de Coimbra, Diploma thesis, 10 2020 5, 41, 108

[Daoud et al. 2020] Daoud, Luka ; Latif, Muhammad K. ; Jacinto, H.S. ; Rafla, Nader : A fully pipelined FPGA accelerator for scale invariant feature transform keypoint descriptor matching. In : *Microprocessors and Microsystems* 72 (2020), pp. 102919. – URL `http://www.sciencedirect.com/science/article/pii/S0141933119300808`. – ISSN 0141-9331 43

[Englehart et Hudgins 2003] Englehart, K. ; Hudgins, B. : A robust, real-time control scheme for multifunction myoelectric control. In : *IEEE Transactions on Biomedical Engineering* 50 (2003), n. 7, pp. 848–854 31

[Ester et al. 1996] Ester, Martin ; Kriegel, Hans-Peter ; Sander, Jörg ; Xu, Xiaowei : A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In : *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1996 (KDD'96), pp. 226–231 112, 118, 119

[Fan et al. 2018] Fan, Hongxiang ; Liu, Shuanglong ; Ferianc, Martin ; Ng, Ho-Cheung ; Que, Zhiqiang ; Liu, Shen ; Niu, Xinyu ; Luk, Wayne : A Real-Time Object Detection Accelerator with Compressed SSDLite on FPGA. In : *2018 International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 14–21 51

[Farina et al. 2021] Farina, Dario ; Vujaklija, Ivan ; Brånemark, Rickard ; Bull, Anthony ; Dietl, Hans ; Graimann, Bernhard ; Hargrove, Levi ; Hoffmann, Klaus-Peter ; Huang, He ; Ingvarsson, Thorvaldur ; Janusson, Hilmar ; Kristjánsson, Kristleifur ; Kuiken, Todd ; Micera, Silvestro ; Stieglitz, Thomas ; Sturma, Agnes ; Tyler, Dustin ; Weir, Richard ; Aszmann, Oskar : Toward higher-performance bionic limbs for wider clinical use. In : *Nature Biomedical Engineering* (2021), 05, pp. 1–13 29

[Fejér et al. 2021a] Fejér, A. ; Nagy, Z. ; Benois-Pineau, J. ; Szolgay, P. ; Rugy, A. de ; Domenger, J-P. : Implementation of Scale Invariant Feature Transform detector

on FPGA for low-power wearable devices for prostheses control. In : *Int J Circ Theor Appl* 49 (2021), pp. 2255 – 2273. – URL https://doi.org/10.1002/cta.3025 26, 27, 80, 110, 146, 151, 152, 153

[Fejér et al. 2019] Fejér, Attila ; Nagy, Zoltán ; Benois-Pineau, Jenny ; Szolgay, Péter ; Rugy, Aymar de ; Domenger, Jean-Philippe : FPGA-based SIFT implementation for wearable computing. In : *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2019, pp. 1–4 26, 37

[Fejér et al. 2021b] Fejér, Attila ; Nagy, Zoltán ; Benois-Pineau, Jenny ; Szolgay, Péter ; Rugy, Aymar de ; Domenger, Jean-Philippe : Array computing based system for visual servoing of neuroprosthesis of upper limbs. In : *2021 17th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, 2021, pp. 1–5 26, 37

[Fejér et al. 2022] Fejér, Attila ; Nagy, Zoltán ; Benois-Pineau, Jenny ; Szolgay, Péter ; Rugy, Aymar de ; Domenger, Jean-Philippe : Hybrid FPGA-CPU-Based Architecture for Object Recognition in Visual Servoing of Arm Prosthesis. In : *Journal of Imaging* 8 (2022), n. 2. – URL https://www.mdpi.com/2313-433X/8/2/44. – ISSN 2313-433X 26, 27, 133, 147, 148, 149, 150, 151

[Fischler et Bolles 1981] Fischler, Martin A. ; Bolles, Robert C. : Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In : *Commun. ACM* 24 (1981), jun, n. 6, pp. 381–395. – URL https://doi.org/10.1145/358669.358692. – ISSN 0001-0782 33, 112, 117

[Friedman et al. 1977] Friedman, Jerome H. ; Bentley, Jon L. ; Finkel, Raphael A. : An algorithm for finding best matches in logarithmic expected time. In : *ACM Transactions on Mathematical Software* (1977) 115

[Fukuda et al. 2021] Fukuda, Osamu ; Sakaguchi, Daisuke ; He, Yunan ; Yamaguchi,

Nobuhiko ; Okumura, Hiroshi : Bimodal Control of a Vision-Based Myoelectric Hand. In : *IEEE Access* 9 (2021), pp. 98369–98380 1, 31, 32, 47

[Gers et al. 2000] Gers, Felix A. ; Schmidhuber, Jürgen ; Cummins, Fred : Learning to Forget: Continual Prediction with LSTM. In : *Neural Computation* 12 (2000), 10, n. 10, pp. 2451–2471. – URL https://doi.org/10.1162/089976600300015015. – ISSN 0899-7667 58

[Ginés et al. 2020] Ginés, Doménech-Asensi ; Juan, Zapata-Pérez ; Ramón, Ruiz-Merino ; Alejandro, López-Alcantud J. ; Ángel, Díaz-Madrid J. ; Manuel, Brea V. ; Paula, López : All-hardware SIFT implementation for real-time VGA images feature extraction. In : *Journal of Real-Time Image Processing* 17 (2020), Apr, n. 2, pp. 371–382. – URL https://doi.org/10.1007/s11554-018-0781-0. – ISSN 1861-8219 40, 42, 109

[Girshick 2015] Girshick, Ross : Fast R-CNN. In : *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448 27, 136, 139, 140

[González-Díaz et al. 2019] González-Díaz, Iván ; Benois-Pineau, Jenny ; Domenger, Jean-Philippe ; Cattaert, Daniel ; de Rugy, Aymar : Perceptually-guided deep neural networks for ego-action prediction: Object grasping. In : *Pattern Recognition* 88 (2019), pp. 223–235. – URL https://www.sciencedirect.com/science/article/pii/S0031320318304011. – ISSN 0031-3203 1, 24, 26, 27, 55, 56, 57, 58, 110, 133, 134, 135, 136, 139, 140, 150

[Han et al. 2020] Han, M. ; Günay, S. Y. ; Schirner, G. ; Padir, T. ; Erdoğmuş, D. : HANDS: a multimodal dataset for modeling toward human grasp intent inference in prosthetic hands. In : *Intell Serv Robot* 13 (2020), n. 1, pp. 179–185 24

[Han et Oruklu 2014] Han, Yan ; Oruklu, Erdal : Real-time traffic sign recognition based on Zynq FPGA and ARM SoCs. In : *IEEE International Conference on Electro/Information Technology*, 2014, pp. 373–376 46, 47, 77

[He et al. 2016] He, K. ; Zhang, X. ; Ren, S. ; Sun, J. : Deep Residual Learning for Image Recognition. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778 27, 30, 34, 35, 36, 48, 57, 76, 137, 138, 146, 147, 150

[Hess 2010] Hess, Rob : An Open-Source SIFTLibrary. In : *Proceedings of the 18th ACM International Conference on Multimedia*. New York, NY, USA : Association for Computing Machinery, 2010 (MM '10), pp. 1493–1496. – URL https://doi.org/10.1145/1873951.1874256. – ISBN 9781605589336 2, 4, 5, 27, 40, 41, 91, 92, 97, 99, 100, 101, 102, 103, 107, 108

[Hochreiter et Schmidhuber 1997] Hochreiter, Sepp ; Schmidhuber, Jürgen : Long Short-Term Memory. In : *Neural Computation* 9 (1997), 11, n. 8, pp. 1735–1780. – URL https://doi.org/10.1162/neco.1997.9.8.1735. – ISSN 0899-7667 58

[Howard et al. 2019] Howard, Andrew ; Sandler, Mark ; Chu, Grace ; Chen, Liang-Chieh ; Chen, Bo ; Tan, Mingxing ; Wang, Weijun ; Zhu, Yukun ; Pang, Ruoming ; Vasudevan, Vijay ; Le, Quoc V. ; Adam, Hartwig : *Searching for MobileNetV3*. 2019. – URL https://arxiv.org/abs/1905.02244 45

[Howard et al. 2017] Howard, Andrew G. ; Zhu, Menglong ; Chen, Bo ; Kalenichenko, Dmitry ; Wang, Weijun ; Weyand, Tobias ; Andreetto, Marco ; Adam, Hartwig : *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017 30, 34, 35, 36, 45, 48

[Huang et al. 2017] Huang, Jonathan ; Rathod, Vivek ; Sun, Chen ; Zhu, Menglong ; Korattikara, Anoop ; Fathi, Alireza ; Fischer, Ian ; Wojna, Zbigniew ; Song, Yang ; Guadarrama, Sergio ; Murphy, Kevin : *Speed/accuracy trade-offs for modern convolutional object detectors*. 2017 36, 48

[Hussein 2015] Hussein, M. T. : A review on vision-based control of flexible manipulators. In : *Advanced Robotics* 29 (2015), n. 24, pp. 1575–1585 24

[Iandola et al. 2014] Iandola, Forrest ; Moskewicz, Matt ; Karayev, Sergey ; Girshick, Ross ; Darrell, Trevor ; Keutzer, Kurt : *DenseNet: Implementing Efficient ConvNet Descriptor Pyramids.* 2014. – URL `https://arxiv.org/abs/1404.1869` 45

[Iandola et al. 2016] Iandola, Forrest N. ; Han, Song ; Moskewicz, Matthew W. ; Ashraf, Khalid ; Dally, William J. ; Keutzer, Kurt : *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.* 2016. – URL `https://arxiv.org/abs/1602.07360` 45

[Intel 2017] Intel : *Intel i5 7300HQ.* 2017. – URL `https://ark.intel.com/content/www/us/en/ark/products/97456/intel-core-i57300hq-processor-6m-cache-up-to-3-50-ghz.html`. – accessed on: 2020-01-14 75, 80, 128, 146

[Kalman 1960] Kalman, Rudolph E. : A new approach to linear filtering and prediction problems. (1960) 33

[Kanishka Madusanka et al. 2017] Kanishka Madusanka, D. G. ; Gopura, R. A. R. C. ; Amarasinghe, Y. W. R. ; Mann, G. K. I. : Hybrid Vision Based Reach-to-Grasp Task Planning Method for Trans-Humeral Prostheses. In : *in IEEE Access* 5 (2017), pp. 16149–16161 24

[Karami et al. 2017] Karami, Ebrahim ; Prasad, Siva ; Shehata, Mohamed S. : Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images. In : *CoRR* abs/1710.02726 (2017). – URL `http://arxiv.org/abs/1710.02726` 40, 48, 83

[Karrenbach et al. 2022] Karrenbach, Maxim ; Boe, David ; Sie, Astrini ; Bennett, Rob ; Rombokas, Eric : Improving Automatic Control of Upper-Limb Prosthesis Wrists Using Gaze-Centered Eye Tracking and Deep Learning. In : *IEEE Trans Neural Syst Rehabil Eng* 30 (2022), February, pp. 340–349 29, 30, 33

[Kathail 2020] Kathail, Vinod : Xilinx Vitis Unified Software Platform. In : *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA : Association for Computing Machinery, 2020 (FPGA '20), pp. 173–174. – URL https://doi.org/10.1145/3373087.3375887. – ISBN 9781450370998 30, 37, 39, 48, 49, 72

[Kramida 2016] Kramida, Gregory : Resolving the Vergence-Accommodation Conflict in Head-Mounted Displays. In : *IEEE Transactions on Visualization and Computer Graphics* 22 (2016), n. 7, pp. 1912–1931 33

[Krausz et al. 2020] Krausz, Nili E. ; Lamotte, Denys ; Batzianoulis, Iason ; Hargrove, Levi J. ; Micera, Silvestro ; Billard, Aude : Intent Prediction Based on Biomechanical Coordination of EMG and Vision-Filtered Gaze for End-Point Control of an Arm Prosthesis. In : *IEEE Trans Neural Syst Rehabil Eng* 28 (2020), May, n. 6, pp. 1471–1480 29, 32, 50

[Krizhevsky et al. 2017] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E. : ImageNet Classification with Deep Convolutional Neural Networks. In : *Commun. ACM* 60 (2017), may, n. 6, pp. 84–90. – URL https://doi.org/10.1145/3065386. – ISSN 0001-0782 34

[Kóta et al. 2019] Kóta, Fülöp ; Zsedrovits, Tamás ; Nagy, Zoltán : Sense-and-avoid system development on an FPGA. In : *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 575–579 46, 47, 77

[LaBRI 2016] LaBRI : *Grasping In The Wild*. 2016. – URL https://www.labri.fr/projet/AIV/dossierSiteRoBioVis/GraspingInTheWildV2.htm. – accessed on: 2020-01-14 1, 36, 48, 51, 58, 60, 61, 62, 90, 97, 115, 130, 141

[Lambooij et al. 2009] Lambooij, Marc ; Ijsselsteijn, Wijnand ; Fortuin, Marten ; Heynderickx, Ingrid : Visual Discomfort and Visual Fatigue of Stereoscopic Dis-

plays: A Review. In : *Journal of Imaging Science and Technology - J IMAGING SCI TECHNOL* 53 (2009), 05 33

[Li et al. 2017] Li, Z. ; Jia, H. ; Zhang, Y. : HartSift: A High-Accuracy and Real-Time SIFT Based on GPU. In : *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, 2017, pp. 135–142 40, 41

[Lin et al. 2016] Lin, Tsung-Yi ; Dollár, Piotr ; Girshick, Ross ; He, Kaiming ; Hariharan, Bharath ; Belongie, Serge : *Feature Pyramid Networks for Object Detection.* 2016. – URL `https://arxiv.org/abs/1612.03144` 57, 139

[Liu et al. 2016] Liu, Wei ; Anguelov, Dragomir ; Erhan, Dumitru ; Szegedy, Christian ; Reed, Scott ; Fu, Cheng-Yang ; Berg, Alexander C. : SSD: Single Shot MultiBox Detector. In : *Lecture Notes in Computer Science* (2016), pp. 21–37. – URL `http://dx.doi.org/10.1007/978-3-319-46448-0_2`. – ISBN 9783319464480 30, 34, 36, 48, 134, 145

[Lowe 2004] Lowe, David G. : Distinctive Image Features from Scale-Invariant Keypoints. In : *International Journal of Computer Vision* 60 (2004), Nov, n. 2, pp. 91–110. – URL `https://doi.org/10.1023/B:VISI.0000029664.99615.94`. – ISSN 1573-1405 1, 2, 16, 26, 37, 40, 41, 42, 48, 76, 83, 84, 85, 87, 88, 94, 95, 111, 112, 113, 116, 125, 132, 133, 146, 150, 151

[Mereu et al. 2021] Mereu, Federico ; Leone, Francesca ; Gentile, Cosimo ; Cordella, Francesca ; Gruppioni, Emanuele ; Zollo, Loredana : Control Strategies and Performance Assessment of Upper-Limb TMR Prostheses: A Review. In : *Sensors* 21 (2021), n. 6. – URL `https://www.mdpi.com/1424-8220/21/6/1953`. – ISSN 1424-8220 29

[Miall et Jackson 2006] Miall, R C. ; Jackson, J K. : Adaptation to visual feedback delays in manual tracking: evidence against the Smith Predictor model of human visually guided action. In : *Exp. Brain Res.* 172 (2006), 06, n. 1, pp. 77–84 25

[Mick et al. 2021] Mick, Sébastien ; Segas, Effie ; Dure, Lucas ; Halgand, Christophe ; Benois-Pineau, Jenny ; Loeb, Gerald E. ; Cattaert, Daniel ; Rugy, Aymar de : Shoulder kinematics plus contextual target information enable control of multiple distal joints of a simulated prosthetic arm and hand. In : *Journal of NeuroEngineering and Rehabilitation* 18 (2021), January, n. 1. – URL `https://doi.org/10.1186/s12984-020-00793-0` 24, 25, 29, 33, 34

[Mick et al. 2019] Mick, Sébastien ; Lapeyre, Mattieu ; Rouanet, Pierre ; Halgand, Christophe ; Benois-Pineau, Jenny ; Paclet, Florent ; Cattaert, Daniel ; Oudeyer, Pierre-Yves ; Rugy, Aymar de : Reachy, a 3D-Printed Human-Like Robotic Arm as a Testbed for Human-Robot Control Strategies. In : *Frontiers in Neurorobotics* 13 (2019), pp. 65. – URL `https://www.frontiersin.org/article/10.3389/fnbot.2019.00065`. – ISSN 1662-5218 25, 55

[Moreau et al. 2019] Moreau, Thierry ; Chen, Tianqi ; Vega, Luis ; Roesch, Jared ; Yan, Eddie ; Zheng, Lianmin ; Fromm, Josh ; Jiang, Ziheng ; Ceze, Luis ; Guestrin, Carlos ; Krishnamurthy, Arvind : A Hardware–Software Blueprint for Flexible Deep Learning Specialization. In : *IEEE Micro* 39 (2019), n. 5, pp. 8–16 30, 37, 39, 79

[Mouchoux et al. 2021] Mouchoux, Jeremy ; Carisi, Stefano ; Dosen, Strahinja ; Farina, Dario ; Schilling, Arndt F. ; Markovic, Marko : Artificial Perception and Semiautonomous Control in Myoelectric Hand Prostheses Increases Performance and Decreases Effort. In : *IEEE Transactions on Robotics* 37 (2021), n. 4, pp. 1298–1312 31, 47

[Muja et Lowe 2009] Muja, Marius ; Lowe, David G. : Fast approximate nearest neighbors with automatic algorithm configuration. In : *In VISAPP International Conference on Computer Vision Theory and Applications*, 2009, pp. 331–340 27, 33, 112, 114, 115, 116, 125, 126

[Ortiz-Catalan et al. 2015] Ortiz-Catalan, Max ; Rouhani, Faezeh ; Branemark,

Rickard ; HAKANSSON, Bo : Offline accuracy: A potentially misleading metric in myoelectric pattern recognition for prosthetic control. In : *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, IEEE, August 2015, pp. 1140–1143 50

[Pablo et al. 2018] PABLO, Rubio-Ibáñez ; RAMÓN, Ruiz-Merino ; GINÉS, Doménech-Asensi ; JAVIER, Martínez-Álvarez J. ; JUAN, Zapata-Pérez ; ÁNGEL, Díaz-Madrid J. ; ALEJANDRO, López-Alcantud J. : An all-hardware implementation of the subpixel refinement stage in SIFT algorithm. In : *International Journal of Circuit Theory and Applications* 46 (2018), n. 9, pp. 1690–1702. – URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2482` 40, 42, 109

[Pappalardo 2021] PAPPALARDO, Alessandro : Xilinx/brevitas. (2021). – URL `https://doi.org/10.5281/zenodo.3333552` 30, 37

[Park et al. 2014] PARK, Jincheol ; LEE, Sanghoon ; BOVIK, Alan C. : 3D Visual Discomfort Prediction: Vergence, Foveation, and the Physiological Optics of Accommodation. In : *IEEE Journal of Selected Topics in Signal Processing* 8 (2014), n. 3, pp. 415–427 33

[Parker et al. 2006] PARKER, P. ; ENGLEHART, K. ; HUDGINS, B. : Myoelectric signal processing for control of powered limb prostheses. In : *Journal of Electromyography and Kinesiology* 16 (2006), n. 6, pp. 541–548. – URL `https://www.sciencedirect.com/science/article/pii/S1050641106001027`. – Special Section (pp. 541–610): 2006 ISEK Congress. – ISSN 1050-6411 29

[Paszke et al. 2019] PASZKE, Adam ; GROSS, Sam ; MASSA, Francisco ; LERER, Adam ; BRADBURY, James ; CHANAN, Gregory ; KILLEEN, Trevor ; LIN, Zeming ; GIMELSHEIN, Natalia ; ANTIGA, Luca ; DESMAISON, Alban ; KÖPF, Andreas ; YANG, Edward ; DEVITO, Zach ; RAISON, Martin ; TEJANI, Alykhan ; CHILAMKURTHY, Sasank ; STEINER,

Benoit ; Fang, Lu ; Bai, Junjie ; Chintala, Soumith : *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* 2019 138, 141

[Pedregosa et al. 2011] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E. : Scikit-learn: Machine Learning in Python. In : *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 27, 112, 118, 119, 121

[Poursanidis et al. 2020] Poursanidis, Miltiadis ; Benois-Pineau, Jenny ; Zemmari, Akka ; Mansenca, Boris ; Rugy, Aymar de : *Move-to-Data: A new Continual Learning approach with Deep CNNs, Application for image-class recognition.* 2020. – URL `https://arxiv.org/abs/2006.07152` 26, 153

[Qasaimeh et al. 2019] Qasaimeh, Murad ; Denolf, Kristof ; Lo, Jack ; Vissers, Kees ; Zambreno, Joseph ; Jones, Phillip H. : Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels. In : *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–8 25, 44, 45, 47

[Redmon et al. 2016] Redmon, Joseph ; Divvala, Santosh ; Girshick, Ross ; Farhadi, Ali : You Only Look Once: Unified, Real-Time Object Detection. (2016). – URL `https://arxiv.org/abs/1506.02640` 30, 34, 36, 48, 134

[Redmon et Farhadi 2016] Redmon, Joseph ; Farhadi, Ali : *YOLO9000: Better, Faster, Stronger.* 2016 36, 48, 145

[Ren et al. 2017] Ren, S. ; He, K. ; Girshick, R. ; Sun, J. : Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), n. 6, pp. 1137–1149. – URL `https://doi.ieeecomputersociety.org/10.1109/TPAMI.2016.2577031` 30, 34, 35, 36, 48, 58, 76, 134, 146, 147, 150

[Rodríguez-Vázquez et al. 2009] Rodríguez-Vázquez, Angel ; Domínguez-Castro, Rafael ; Jiménez-Garrido, Francisco ; Morillas, Sergio ; García, Alberto ; Utrera, Cayetana ; Pardo, Ma. D. ; Listan, Juan ; Romay, Rafael : A CMOS Vision System On-Chip with Multi-Core, Cellular Sensory-Processing Front-End. In : Roska, Tamás (Ed.) ; Baatar, C. (Ed.) ; Porod, W. (Ed.): *Cellular Nanoscale Sensory Wave Computing*. Springer US, oct 2009, pp. 129–146 41, 42, 108

[Rosten et Drummond 2006] Rosten, Edward ; Drummond, Tom : Machine Learning for High-Speed Corner Detection. In : Leonardis, Aleš (Ed.) ; Bischof, Horst (Ed.) ; Pinz, Axel (Ed.): *Computer Vision – ECCV 2006*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006, pp. 430–443. – ISBN 978-3-540-33833-8 40

[Rousseeuw 1984] Rousseeuw, Peter J. : Least median of squares regression. In : *Journal of the American Statistical Association* (1984), pp. 871–880 117

[Rublee et al. 2011] Rublee, Ethan ; Rabaud, Vincent ; Konolige, Kurt ; Bradski, Gary : ORB: An Efficient Alternative to SIFT or SURF. In : *Proceedings of the 2011 International Conference on Computer Vision*. Washington, DC, USA : IEEE Computer Society, 2011, pp. 2564–2571. – URL `http://dx.doi.org/10.1109/ICCV.2011.6126544`. – ISBN 978-1-4577-1101-5 33, 40, 48, 83

[de San Roman et al. 2017] San Roman, Philippe P. de ; Benois-Pineau, Jenny ; Domenger, Jean-Philippe ; Paclet, Florent ; Cattaert, Daniel ; Rugy, Aymar de : Saliency Driven Object recognition in egocentric videos with deep CNN: toward application in assistance to Neuroprostheses. In : *Comput. Vis. Image Underst.* 164 (2017), pp. 82–91 25

[Sandler et al. 2018] Sandler, Mark ; Howard, Andrew ; Zhu, Menglong ; Zhmoginov, Andrey ; Chen, Liang-Chieh : MobileNetV2: Inverted Residuals and Linear Bottlenecks. (2018). – URL `https://arxiv.org/abs/1801.04381` 45

[Scott 2016] Scott, Stephen H. : A functional taxonomy of bottom-up sensory feedback processing for motor actions. In : *Trends Neurosci.* 39 (2016), n. 8, pp. 512–526 25

[Shao et al. 2015] Shao, A-jun ; Wei-xian, Qian ; Guo-hua, Gu ; Kai-li, Lu : Real-time implementation of SIFT feature extraction algorithms in FPGA. In : Shi, Guangming (Ed.) ; Li, Xuelong (Ed.) ; Huang, Bormin (Ed.): *2015 International Conference on Optical Instruments and Technology: Optoelectronic Imaging and Processing Technology* Proc. SPIE 9622 International Society for Optics and Photonics (Organizer), SPIE, 2015, pp. 233 – 247. – URL `https://doi.org/10.1117/12.2190330` 40, 43, 89, 91, 109

[Shi et al. 2022a] Shi, Chunyuan ; Yang, Dapeng ; Qiu, Siyang ; Zhao, Jingdong : *i-GSI: A Fast and Reliable Grasp-type Switching Interface based on Augmented Reality and Eye-tracking.* 2022. – URL `https://arxiv.org/abs/2204.10664` 29, 32, 33

[Shi et al. 2022b] Shi, Chunyuan ; Yang, Dapeng ; Zhao, Jingdong ; Jiang, Li : *i-MYO: A Hybrid Prosthetic Hand Control System based on Eye-tracking, Augmented Reality and Myoelectric signal.* 2022. – URL `https://arxiv.org/abs/2205.08948` 29, 32, 33

[Silpa-Anan et Hartley 2008] Silpa-Anan, Chanop ; Hartley, Richard : Optimised KD-trees for fast image descriptor matching. In : *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8 115

[Simonyan et Zisserman 2015] Simonyan, Karen ; Zisserman, Andrew : *Very Deep Convolutional Networks for Large-Scale Image Recognition.* 2015 30, 34, 35, 36, 48

[Solymár et al. 2011] Solymár, Zóra ; Stubendek, Attila ; Radványi, Mihály ; Karacs, Kristóf : Banknote recognition for visually impaired. In : *2011 20th European Conference on Circuit Theory and Design (ECCTD)*, 2011, pp. 841–844 83

[Sun et al. 2018] Sun, Yongping ; Yang, Ming ; Chen, Yangyang ; He, Wangpin ; Xu, Dianguo : An SoC-based platform for integrated multi-axis motion control and motor drive. In : *2018 International Power Electronics Conference (IPEC-Niigata 2018 -ECCE Asia)*, 2018, pp. 560–564 46, 47

[Suárez et al. 2014] Suárez, M. ; Brea, V. M. ; Fernández-Berni, J. ; Carmona-Galán, R. ; Cabello, D. ; Rodríguez-Vázquez, A. : Gaussian pyramid extraction with a CMOS vision sensor. In : *2014 14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, July 2014, pp. 1–2. – ISSN 2165-0152 41, 42

[Szegedy et al. 2015] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew : Going deeper with convolutions. In : *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9 34

[Szeliski 2011] Szeliski, Richard : *Computer Vision*. Springer London, 2011. – URL `https://doi.org/10.1007/978-1-84882-935-0` 116

[Toshiba 2019] Toshiba : *SPS*. 10 2019. – URL `http://www.toshiba-teli.co.jp/en/products/industrial/sps/sps.htm`. – accessed on: 2019-03-26 42

[Trimberger 2015] Trimberger, Stephen M. : Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. In : *Proceedings of the IEEE* 103 (2015), n. 3, pp. 318–331 64

[Umuroglu et al. 2017] Umuroglu, Yaman ; Fraser, Nicholas J. ; Gambardella, Giulio ; Blott, Michaela ; Leong, Philip ; Jahre, Magnus ; Vissers, Kees : FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In : *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017), Feb. – URL `http://dx.doi.org/10.1145/3020078.3021744` 30, 37, 38

[Vourvoulakis et al. 2016] Vourvoulakis, John ; Kalomiros, John ; Lygouras, John : Fully pipelined FPGA-based architecture for real-time SIFT extraction. In : *Microprocessors and Microsystems* 40 (2016), pp. 53–73. – URL https://www.sciencedirect.com/science/article/pii/S0141933115001921. – ISSN 0141-9331 40, 42, 89, 109

[Vourvoulakis et al. 2017] Vourvoulakis, John ; Kalomiros, John ; Lygouras, John : FPGA accelerator for real-time SIFT matching with RANSAC support. In : *Microprocessors and Microsystems* 49 (2017), pp. 105 – 116. – URL http://www.sciencedirect.com/science/article/pii/S0141933116303623. – ISSN 0141-9331 40, 42, 89, 109

[Vörösházi et al. 2008] Vörösházi, Zsolt ; Kiss, András ; Nagy, Zoltán ; Szolgay, Péter : Implementation of embedded emulated-digital CNN-UM global analogic programming unit on FPGA and its application. In : *International Journal of Circuit Theory and Applications* 36 (2008), n. 5-6, pp. 589–603. – URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.507 92

[Xilinx 2017] Xilinx : *Vivado AXI Reference Guide*. 04 2017. – URL https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf. – accessed on: 2020-01-14 68, 90

[Xilinx 2018a] Xilinx : *PYNQ*. 2018. – URL http://www.pynq.io/home.html. – accessed on: 2019-01-14 70

[Xilinx 2018b] Xilinx : *Vivado Design Tools*. 2018. – URL https://www.xilinx.com/products/design-tools/vivado.html. – accessed on: 2021-02-04 5, 98, 105, 106, 109

[Xilinx 2019] Xilinx : *UG1182 ZCU102 Evaluation Board - User Guide*. 6 2019. – URL https://www.xilinx.com/support/documentation/boards_and_kits/zcu102/ug1182-zcu102-eval-bd.pdf. – accessed on: 2022-07-16 45, 65, 78, 133, 146, 150

[Xilinx 2021a] Xilinx : *DS891 - Zynq UltraScale+ MPSoC Data Sheet: Overview*. 5 2021. – URL `https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf`. – accessed on: 2020-01-14 75, 77, 80, 128, 129, 133, 146, 150

[Xilinx 2021b] Xilinx : *PG338 - Zynq DPU v3.3 IP Product Guide (v3.3)*. 2 2021. – URL `https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_3/pg338-dpu.pdf`. – accessed on: 2022-07-16 79

[Xilinx 2021c] Xilinx : *UG1431 (v1.4): Vitis Ai documentation*. 07 2021. – URL `https://docs.xilinx.com/v/u/1.4-English/ug1431-vitis-ai-documentation`. – accessed on: 2022-01-20 145

[Xilinx 2022a] Xilinx : *UG1075 (v1.11): Zynq UltraScale+ Device Packaging and Pinouts*. 1 2022. – URL `https://docs.xilinx.com/v/u/en-US/ug1075-zynq-ultrascale-pkg-pinout`. – accessed on: 2022-07-16 67

[Xilinx 2022b] Xilinx : *UG1399 (v2022.1): Vitis HLS User Guide*. 1 2022. – URL `https://docs.xilinx.com/r/en-US/ug1399-vitis-hls`. – accessed on: 2022-07-16 68

[Xilinx 2022c] Xilinx : *UG1414 (v2.5): Vitis AI User Guide*. 6 2022. – URL `https://docs.xilinx.com/r/en-US/ug1414-vitis-ai`. – accessed on: 2022-07-16 1, 72, 73, 74

[Yu et al. 2020] Yu, Yunxuan ; Zhao, Tiandong ; Wang, Kun ; He, Lei : Light-OPU: An FPGA-Based Overlay Processor for Lightweight Convolutional Neural Networks. In : *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. New York, NY, USA : Association for Computing Machinery, 2020 (FPGA '20), pp. 122–132. – URL `https://doi.org/10.1145/3373087.3375311`. – ISBN 9781450370998 25, 44, 45, 47

[Zarándy et al. 2016] Zarándy, Ákos ; Nemeth, Mate ; Nagy, Zoltan ; Kiss, Andras ; Santha, Levente ; Zsedrovits, Tamás : A real-time multi-camera vision system for UAV collision warning and navigation. In : *Journal of Real-Time Image Processing* 12 (2016), Dec, n. 4, pp. 709–724. – URL https://doi.org/10.1007/s11554-014-0449-3. – ISSN 1861-8219 51

[Zarándy et al. 2016] Zarándy, Ákos ; Nemeth, Mate ; Nagy, Zoltan ; Kiss, Andras ; Santha, Levente ; Zsedrovits, Tamás : A Real-time Multi-camera Vision System for UAV Collision Warning and Navigation. In : *J. Real-Time Image Process.* 12 (2016), December, n. 4, pp. 709–724. – URL https://doi.org/10.1007/s11554-014-0449-3. – ISSN 1861-8200 83

[Zhang et al. 2017] Zhang, Xiangyu ; Zhou, Xinyu ; Lin, Mengxiao ; Sun, Jian : *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices.* 2017. – URL https://arxiv.org/abs/1707.01083 45

[Zhu et al. 2021] Zhu, Xizhou ; Su, Weijie ; Lu, Lewei ; Li, Bin ; Wang, Xiaogang ; Dai, Jifeng : *Deformable DETR: Deformable Transformers for End-to-End Object Detection.* 2021 26

Appendix A

Publications related to the thesis

A.1 Journal publications

- **Fejér, A.** ; Nagy, Z. ; Benois-Pineau, J. ; Szolgay, P. ; Rugy, A. de ; Domenger, J-P. : Implementation of Scale Invariant Feature Transform detector on FPGA for low-power wearable devices for prostheses control. In : *Int J Circ Theor Appl 49* (2021), pp. 2255 – 2273. – URL `https://doi.org/10.1002/cta.3025`

- **Fejér, Attila** ; Nagy, Zoltán ; Benois-Pineau, Jenny ; Szolgay, Péter ; Rugy, Aymar de ; Domenger, Jean-Philippe : Hybrid FPGA-CPU-Based Architecture for Object Recognition in Visual Servoing of Arm Prosthesis. In : *Journal of Imaging 8* (2022), n. 2. – URL `https://www.mdpi.com/2313-433X/8/2/44`. – ISSN 2313-433X

A.2 Conference publications

- **Fejér, Attila** ; Nagy, Zoltán ; Benois-Pineau, Jenny ; Szolgay, Péter ; Rugy, Aymar de ; Domenger, Jean-Philippe : FPGA-based SIFT implementation for wearable computing. In : *2019 IEEE 22nd International Symposium on*

Design and Diagnostics of Electronic Circuits Systems (DDECS), 2019, pp. 1-4

- **Fejér, Attila** ; Nagy, Zoltán ; Benois-Pineau, Jenny ; Szolgay, Péter ; Rugy, Aymar de ; Domenger, Jean-Philippe : Array computing based system for visual servoing of neuroprosthesis of upper limbs. In : *2021 17th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, 2021, pp. 1–5