Universidade do Minho
Escola de Engenharia

João Pedro Borges Araújo Oliveira e Silva

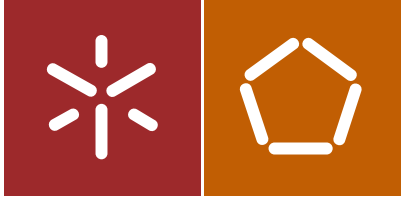
Development of Human Body Pose Detection Algorithms for In-Car Scenario, and Validation with Suitable Ground-Truth System

May, 2020

Development of Human Body Pose Detection Algorithms for In-Car Scenario, and Validation with Suitable Ground-Truth System

João Pedro Borges Araújo Oliveira e Silva

UMinho | 2020



Universidade do Minho
Escola de Engenharia

João Pedro Borges Araújo Oliveira e Silva

Development of Human Body Pose Detection Algorithms for In-Car Scenario, and Validation with Suitable Ground-Truth System

Doctoral Thesis in Electronics and Computers Engineering

Trabalho efetuado sob a orientação do
Jaime Francisco Cruz Fonseca

May, 2020

Acknowledgments

Começo por agradecer aqueles que me são mais importantes, a minha família. Catarina, companheira, confidente, fonte de energia e inspiração. Aos meus pais, José e Maria, por serem uma fonte basilar do meu ser, por me terem moldado na pessoa que sou hoje, e por mais uma vez serem um agente importante numa etapa da minha vida, o meu Doutoramento. Às minhas irmãs, Maria e Ana, e cunhado Jorge que sempre foram e sempre serão uma fonte de apoio e inspiração em todas as fases da minha vida. Sem vós não teria conseguido ultrapassar esta nova etapa. Ao meu afilhado Henrique e sobrinho Afonso, são e serão a minha fonte de inspiração, e para sempre o meu destino será moldado ao vosso. Quero agradecer aos meus avós António e Aurora e aos meus tios Toni e Gaspar, que já não estando aqui comigo, sei que estarão sempre a meu lado. Aos meus padrinhos, José e Rosa, à minha tia Maria da Glória, aos meus primos Manuel, Mafalda, Mariana e Nuno. Aos meus sogros Lima e Sameiro, e cunhados Abilio e Sofia. A ti Mina, um obrigado por estares sempre presente, disponivel, alegre em todos os momentos que mais precisei.

Quero agradecer ao meu orientador, Doutor Jaime Cruz Fonseca, por me ter acompanhado neste Doutoramento, por me ter apoiado em todos os momentos de dificuldade. Pessoa cuja partilha vai para além da formalidade, conheci como professor mas hoje tenho como amigo. Ao professor José Mendes por ter acreditado nas minhas capacidades e me ter dado autonomia de gestão e decisão ao longo de todo o projeto INNOVCAR: DMC2.0. Pelo seu apoio nos momentos mais criticos.

I would also like to thank my colleagues, Helena, Bruno, Nelson, Sandro, José Brito, Victor Coelho, Johannes Pallauf, Maximilian Schiller, without them i would not be able to fullfill all my Doctoral plan expectactions, due to their friendship, technical skills and empathy, a sincere thank you.

This work is supported by: European Structural and Investment Funds in the FEDER component, through the (COMPETE 2020) [Project nº 002797; Funding Reference: POCI-01-0247-FEDER-002797].

A todos vós, o meu obrigado sincero!

João Pedro

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that i have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that i have fully acknowledged the Code of Ethical Conducted of the University of Minho.

University of Minho, May 29, 2020

Full name: João Pedro Borges Araújo Oliveira e Silva

Signature:

Abstract

Automated driving cars are emerging, increasing the need for advanced occupant monitoring applications. A transversal need for such systems is the detection of the occupants' posture. Discriminative approaches have received increased focus in the past decade, due to its automated detection and the growth in Machine Learning (ML) applications and frameworks. One of its downsides is the need for a large dataset to train, to achieve high accuracy. To allow a robust algorithmic training and validation, an algorithmic development pipeline able to generate both real and synthetic datasets in the in-car scenario needs to be established, together with adequate evaluation procedures, this thesis addresses such development. The approach focuses first in two toolchains for in-car human body pose dataset generation: (1) real, and (2) synthetic. The first toolchain uses two types of sensors for the data generation: (1) image data is captured through a Time-of-Flight (ToF) sensor, and (2) human body pose data (ground-truth) is captured through an inertial suit and optical system. Besides quantifying the inertial suit inherent sensitivity and accuracy, the feasibility of the overall system for human body pose capture in the in-car scenario was demonstrated. Finally, the feasibility of using system generated data (which was made publicly available) to train ML algorithms is demonstrated. The second toolchain uses the features and labels from the previous one, in this case both sensors are synthetically rendered. The toolchain creates a customized synthetic environment, comprising human models, car, and camera. Poses are automatically generated for each human, taking into account a per-joint axis Gaussian or incremental distribution, constrained by anthropometric and Range of Motion measurements. Scene validation is done through collision detection. Rendering is focused on vision data, supporting ToF and RGB cameras, generating synthetic images from these sensors. The feasibility of using synthetic data (which was made publicly available), combined with real data, to train distinct machine learning agorithms is demonstrated. Finally, several algorithms were evaluated, and a Deep Learning (DL) based algorithm, namely Part Affinity Fields, was selected, customized and trained with datasets generated with the previously mentioned toolchains, ultimately aiming to improve accuracy for the in-car scenario.

Resumo

Veículos totalmente autónomos estão a emergir, aumentando a necessidade de um sistema de deteção avançada dos ocupantes. Uma necessidade transversal destes sistemas é a deteção da postura dos ocupantes. Abordagens discriminativas tem gerado um maior interesse na última década, muito devido à sua deteção automática bem como ao aumento de aplicações de Machine Learning (ML).Contudo, estes necessitam de um grande conjunto de dados de treino, de forma a aumentar a precisão. Para permitir um treino e validação robusta, é necessário estabelecer um pipeline de desenvolvimento algoritmico capaz de gerar conjuntos de dados reais e sintéticos para o cenário automóvel, juntamente com procedimentos de avaliação adequados, esta tese visa este desenvolvimento. Esta foca-se inicialmente no desenvolvimento de duas toolchains para a geração de datasets de pose humana no interior do veículo: (1) reais, e (2) sintéticos. A primeira toolchain utiliza dois tipos de sensores para a geração de dados: (1) imagens através de um sensor Time-of-Flight (ToF), e (2) a pose humana (ground-truth) é através de um fato inercial e um sistema óptico. Para além de quantificar a sensibilidade e precisão inerente do sistema inercial, a viabilidade do sistema completo para captura de pose humana no interior do veículo foi demonstrada. Por fim, é demonstrada a viabilidade de usar dados reais (disponibilizados publicamente), para treinar algoritmos ML. A segunda toolchain utiliza as mesmas features e labels da anterior, neste caso ambos os sensores são sintéticos. Esta cria um cenário customizavel, constituído por modelos humanos, carro, e câmera. As poses são geradas automaticamente para cada humano, tendo em conta uma distribuição Gaussiana ou incremental, sendo estas restringidas por medidas antropométricas. Diferentes tipos de deteção de colisões são avaliados de forma a validar os dados, nomeadamente corpo-corpo, humano-humano, e humano-carro. A renderização é focada em câmeras ToF e RGB, gerando imagens sintéticas destes sensores. É demonstrada a viabilidade de utilizar dados sintéticos (disponibilizados publicamente), para treinar algoritmos ML. Finalmente, vários algoritmos foram avaliados, e um algoritmo, nomeadamente Part Affinity Fields, foi selecionado, modificado e treinado com os datasets gerados através das toolchains mencionadas anteriormente, de forma a aumentar a sua precisão para o cenário do interior do veículo.

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

ADTF Automated Data and Time-Triggered Framework. 44, 45, 46, 53, 54, 55, 56, 59, 60, 70, 72, 73, 74, 77, 79, 82

API Application Programming Interface. 7, 14, 29, 36, 39

AUC Area Under Curve. 96, 100, 101, 125, 131, 160, 161, 163, 173

CMU Carnegie Mellon University. 1, 8, 15, 16, 105

CNN Convolutional Neural Network. 9, 17, 96, 119, 125, 150

CPU Central Processing Unit. 12, 39, 42, 75, 114, 122, 123

CW Continuous Wave. 34

DL Deep Learning. v, 9, 10, 22, 23, 47, 135

DoF Degrees-of-Freedom. 3, 7, 12, 25, 27, 28, 29, 31, 32, 105, 106

DPM Deformable Part Models. 135, 136, 137, 141, 143, 146, 147

DT Decision Tree. 137

EN European Standard. 36, 37

FICP Finite Iterative Closest Point. 10, 11, 12, 61, 88

FoV Field-of-View. 16, 37, 41, 106, 111, 112, 131, 150, 153, 154

fps frames per second. 7, 9, 36, 137, 139, 140, 162

GPU Graphics Processing Unit. 7, 8, 13, 39, 42, 43, 122, 162

GUI Graphical User Interface. 48, 53, 114, 115, 116, 166

HoG Histogram of Oriented Gradients. 9, 135, 136

IC Interprocess Communications. 166, 173

IEC International Electrotechnical Commission. 36

IMMUs Inertial and Magnetic Measurement Units. 5, 177, 178

IOU Intersection Over Union. 141

IR Infrared. 36, 37

ISB Internal Society of Biomechanics. 5, 25, 27, 28, 29, 31, 52, 58, 59, 60, 105

ISO International Organization for Standardization. 37

ITOP Towards Viewpoint Invariant 3D Human Pose Estimation. 95, 100, 143, 147

JSON JavaScript Object Notation. 89, 109

KF Kalman Filter. 10, 14, 26

LED Light-Emitting Diode. 34, 37, 38, 40

LSC Local Shape Context. 12

ML Machine Learning. v, 4, 23, 42, 43, 47, 70, 71, 94, 100, 102, 103, 123, 130

NST Neural Style Transfer. xvi, 104, 109, 111, 112, 124, 125, 126, 132, 179, 180

PAF Part Affinity Fields. 21, 23, 96, 125, 135, 141, 142, 143, 147, 164

PF Particle Filter. 10, 12, 14

PM Pulse Modulation. 34, 35, 75

PoV Point-of-View. xi, xxiii, xxiv, 12, 150, 164, 165, 167, 168, 169, 173, 174, 177

PPF Partitioned Particle Filter. 10

PSO Particle Swarm Optimization. 10, 11

RAM Random-Access Memory. 42, 43, 122

RFoC Random Forest Classifier. 7, 8, 9, 10

RoM Range of Motion. 69, 105

RTW Random Tree Walks. 135, 137, 139, 140, 143, 146, 147

SAV Shared Autonomous Vehicles. 162, 180

SDK Software Development Kit. 9, 25, 27, 28, 29, 30, 31, 32, 36, 39, 41, 51, 55, 59, 74, 75, 76, 77, 78, 79, 91

SLERP Spherical Linear Quaterninon Interpolation. 57, 61

SNR Signal-to-Noise Ratio. 40, 75

SSD Solid-State Drive. 43, 122

SVM Support Vector Machine. 10, 136

ToF Time-of-Flight. v, xii, xiii, xiv, xxiii, xxiv, xxxiv, 11, 15, 16, 17, 19, 20, 21, 33, 34, 36, 37, 38, 39, 41, 70, 72, 73, 74, 77, 79, 80, 81, 82, 83, 85, 86, 88, 89, 101, 104, 111, 118, 132, 150, 155, 164, 165, 166, 168, 169, 173, 174, 177, 178, 179

U-Net Convolutional Networks for Biomedical Image Segmentation. 126

UDP User Datagram Protocol. 29, 59

YOLO You Only Look Once. 135, 140, 141, 143, 147

List of Symbols

q Quaternion matrix. 26, 54, 58

${}^{G}p_{landmark}$ Positions of anatomical landmarks wrt. the joint origin of the related segment global frame. 26

G Segment global frame. 26

${}^{B}x$ Vector of the landmark. 26

B Body frame. 26

${}^{GB_B}q$ Orientation of distal segment. 26

${}^{GB_A}q$ Orientation of proximal segment. 26

${}^{GB}q_{seg}$ Quaternion vector describing the orientation of a segment wrt. the global frame. 26

ϕ ToF phase shift. 34, 35

$s(t)$ ToF emitted signal. 34, 35

$r(t)$ ToF received signal. 34, 35

d ToF pixel-object distance considered the depth frame. 34, 35, 36, 39

d_{max} ToF pixel-object maximum distance limit due to phase wrapping. 35

lc Light speed constant. 35

$C(\varkappa)$ ToF cross-correlation between emitted and received signals. xii, 34, 35, 36

tA ToF amplitude of the received signal, considered the amplitude frame. 34, 35, 36, 39

tB ToF offset coefficient due to the ambient illumination. 34, 35

f_m ToF modulation frequency. 34, 35, 75

τ ToF time of flight for light sent from the camera and received by it after being reflected by the object. 34, 35

f_{out} EVK75023 output frame rate with temporal filtering. 40

f_{in} EVK75023 output frame rate without temporal filtering. 40, 75

$FIFO_{stacks}$ EVK75023 number of frames for temporal median filtering. 40

IT EVK75123 integration time. 75

$NPhase$ EVK75123 number of phases used for amplitude and depth estimation (2 or 4). 75

$CPUclock$ EVK75123 internal cpu frequency in MHz. 75

$amp_{x,y}$ ToF standard format for a ampltiude frame. 36, 39, 80, 166

$depth_{x,y}$ Standard format for a single depth frame. Blender rendered ToF depth frame. 36, 39, 110, 115, 119, 150, 157, 166

pc_f Standard format for a single point-cloud matrix. Blender rendered ToF 3D point-cloud matrix. 36, 39, 112, 115, 121, 166

f 2D to 1D concatenated pixel indexing. 82, 84

IM_C ToF pre-calibrated intrinsic matrix. 81, 86, 90, 156, 157

f_x ToF pre-calibrated focal length in pixels in x-axis. 81

f_y ToF pre-calibrated focal length in pixels in y-axis. 81

c_x ToF pre-calibrated optical center in pixels in x-axis. 81

c_y ToF pre-calibrated optical center in pixels in y-axis. 81

sk ToF pre-calibrated axes skew angle. 81

λ_t ADTF raw timestamps in microseconds. xii, 45, 47, 55, 57, 58, 74, 77, 84

T ADTF number of recorded samples from all filters. 45

$S_{s,n}$ ADTF data sample containing a set of variables {...}. 45, 54, 55, 58, 59, 74, 77

s ADTF filter indexing. 45, 57, 58

n ADTF filter sample indexing. 57, 82

N ADTF number of recorded samples for a filter. 45

Γ ADTF user data sent to filter. xii, xiv, 44, 54, 74, 77, 78

i ADTF filter segment indexing. 57

$Q_{s,n,i}$ ADTF body segment quaternion data sample. 57, 58

$B_{s,n,i}$ ADTF body segment name data sample. 57

j Human body/model joint indexing. 58, 61, 62, 82, 105, 153

$B_{s,n,j}$ ADTF body joint name data sample. 61

$k_{s,n,j}$ ADTF body joint 3D position data sample. 61

$Q_{s,n,j}$ ADTF body joint quaternion data sample. 61

$\Lambda_{t'}$ Matlab synchronized timestamps in microseconds. 57, 58, 84, 88

t' Matlab or Blender synchronized sample indexing. 58, 62, 82, 89, 115, 116, 144

Γ_f Matlab user defined frequency for synchronization. 57, 58

$\delta_{s,t',i}$ Matlab synchronized data sample containing a set of synchronized variables {...} directly related with $S_{s,n,i}$. 57, 58

$\delta_{s,t',j}$ Matlab synchronized data sample containing a set of synchronized variables {...} directly related with $S_{s,n,j}$. 61, 62

$\upsilon_{s,t',i}$ Matlab synchronized body segment name data sample. 57

$\varphi_{s,t',i}$ Matlab synchronized body joint quaternion data sample. 57, 58

$\upsilon_{s,t',j}$ Matlab synchronized body joint name data sample. 61

$\kappa_{s,t',j}$ Matlab synchronized body joint 3D position data sample. 61, 62

$\varphi_{s,t',j}$ Matlab synchronized body joint quaternion data sample. 61

$s\Theta$ Sensor angle of rotation in euler representation for an individual axis rotation. 52

$m\Theta$ Motor angle of rotation wrt. its rotation axis. 52

$s\mu$ Sensor mean error wrt. an individual axis of rotation. 52

$s\sigma$ Sensor error standard deviation wrt. a axis of rotation. 52

nI Motor rotation iterations. 52

nS Sensor samples in each 360^o rotation. 52, 63

ξ_t Sensor error in each timestamp wrt. an individual axis of rotation. 52

$Pxyz$ 3D cartesian position. 53, 59, 63, 156, 157

nJ Number of joints. 53, 153, 154

T_{pelvis} 4x4 Transformation matrix for the pelvis joint wrt. the MVN Awinda global coordinate system. 53

T_j^{pelvis} 4x4 Transformation matrix for each joint wrt. the pelvis. 53, 61

ξ_j Individual body joint Euclidian error. xiii, 52, 53, 65

θ Angle of rotation in axis-angle representation. 54, 58, 81

$(q_0)^c$ Quaternion conjugate for the first sample used for removal of offset rotation between systems. 54, 81

μ_i MVN Awinda body segment mean error. 58, 65

σ_i MVN Awinda body segment error standard deviation. 58

$\xi_{t',i}$ MVN Awinda body segment error for each timestamp. 58

$M_{s,t'}$ 3D mesh from $\kappa_{s,t',j}$ for each filter where each joint is a xyz vertex position. 61, 62

$T_{\delta_{1,t',j}}^{\delta_{2,t',j}}$ Transformation matrix that best aligns two body pose samples from MVN Awinda and Vicon systems. 62

μ_j MVN Awinda body pose joint mean error. 62, 63, 67

σ_j MVN Awinda body pose joint error standard deviation. 62, 63

$\xi_{t',j}$ MVN Awinda body pose joint Euclidian error for each timestamp. 58, 62, 63

C ToF optical center. xiii, 72

A MVN Awinda head joint. xiii, 72

J MVN Awinda body joints. xiii, 72, 87

W Vicon global coordinate system. xiii, 72

O Subject's head object tracked by the Vicon system. xiii, 72

T_J^C Transformation matrix that aligns MVN Awinda body joints wrt. the ToF global coordinate systemr. 72, 88, 89

P_J^C 3x1 Translation matrix for each joint wrt. the the ToF global coordinate system. 90

T_J^A Transformation matrix that aligns MVN Awinda body joints wrt. the head joint. 87, 88, 89

T_A^O Transformation matrix that aligns MVN Awinda head joint wrt. the subject's head joint in Vicon system. xiv, 88, 89

T_O^W Transformation matrix that aligns the subject's head joint in the Vicon global coordinate system. 84, 88, 89

T_W^C Transformation matrix that aligns the Vicon global coordinate system wrt. the ToF global coordinate system. xiv, xxv, 81, 85, 86, 87, 88, 89

Pxy_C Vicon marker 2D pixel position in the ToF image plane (detected by the ToF camera). 80, 81

Pxy_{marker}^C Vicon marker 2D pixel position in the ToF image plane (detected by the Vicon system). 80, 81

$Pxyz_{marker}^C$ Vicon marker 3D position in the ToF global coordinate system (detected by the Vicon system). 80, 81

$Pxyz_{marker}^W$ Vicon marker 3D position wrt. the Vicon global coordinate system. 79, 81

$t(0)_{Vicon}^{ToF}$ Time delay between Vicon and ToF in microseconds. 81, 83

$t(0)_{Awinda}^{Vicon}$ Time delay between MVN Awinda and Vicon in microseconds. xiv, 81, 82, 83

$t(0)_{Awinda}^{ToF}$ Time delay between MVN Awinda and ToF in microseconds. 83

t_C All timestamps for the ToF sensor $\lambda_t \cap S_{s=tof,n}$. 83

t_O All timestamps for the Vicon system, $\lambda_t \cap S_{s=vicon,n}$. 83

t_A All timestamps for the MVN Awinda system, $\lambda_t \cap S_{s=awinda,n}$. 83

$S_{s=tof,n}$ ADTF ToF data sample containing a set of variables {...}. 82, 84

$S_{s=vicon,n,}$ ADTF Vicon data sample containing a set of variables {...}. 83, 84

$S_{s=awinda,n,j}$ ADTF MVN Awinda data sample containing a set of variables {...}. 82, 84

$AMP_{n,f}$ ADTF ToF amplitude frame sample. 82, 84

$PC_{n,v}$ ADTF ToF 3D point-cloud matrix sample. 82, 84

v ADTF ToF point-cloud matrix 1D concatenated voxel indexing. 82, 84

T_n ADTF Vicon subject's head 3D position data sample. 82, 83, 84

R_n ADTF Vicon subject's head quaternion data sample. 82, 83, 84

$\delta_{s=tof,t'}$ Matlab synchronized data sample containing a set of synchronized variables {...} directly related with $S_{s=tof,n}$. 82

$\delta_{s=vicon,t'}$ Matlab synchronized data sample containing a set of synchronized variables {...} directly related with $S_{s=vicon,n}$. 83, 84

$\delta_{s=awinda,t',j}$ Matlab synchronized data sample containing a set of synchronized variables {...} directly related with $S_{s=awinda,n,j}$. 82, 84

$depth_{t',x,y}$ Standard dataset depth frame format. Matlab synchronized ToF depth frames. 82, 84, 89, 116, 120, 144, 155, 166, 168

$amp_{t',x,y}$ Standard dataset amplitude frame format. Matlab synchronized ToF amplitude frame. xiv, 82, 84, 86, 87, 89, 166

X ToF horizontal resolution. 89, 120

Y ToF vertical resolution. 89, 120

$pc_{t',f}$ Standard dataset 3D point-cloud matrix format. Matlab synchronized ToF 3D point-cloud matrix. 82, 89, 116, 166, 168

$pcx_{t',f}$ Matlab synchronized ToF 3D point-cloud matrix x-coordinates. 82, 84, 89, 90, 116

$pcy_{t',f}$ Matlab synchronized ToF 3D point-cloud matrix y-coordinates. 82, 84, 89, 90, 116

$pcz_{t',f}$ Matlab synchronized ToF 3D point-cloud matrix z-coordinates. 82, 84, 89, 90, 116

$\kappa_{t',j}$ Matlab synchronized MVN Awinda body joints 3D position data sample. 82, 84, 87

$\varphi_{t',j}$ Matlab synchronized MVN Awinda body joints quaternion data sample. 82, 84, 87

$\rho_{t'}$ Matlab synchronized Vicon subject's head 3D position data sample. 82, 83, 84

$\varrho_{t'}$ Matlab synchronized Vicon subject's head quaternion data sample. 82, 83, 84, 88

$\varphi_{t',head}$ Matlab synchronized MVN Awinda head quaternion data sample. 87, 88

$\kappa_{t',head}$ Matlab synchronized MVN Awinda head joint 3D position data sample. 87

T_B^C Transformation matrix that aligns the checkerboard plane wrt. the ToF global coordinate system. 85, 86

P_B^C 3x1 matrix with the 3D position of the set of markers wrt. the ToF global coordinate system. 85, 86

p_B^C 3x1 matrix with the origin coordinate system of the ToF global coordinate system. 85, 86

μP_B^C 3x1 matrix with the centroid of the set of markers wrt. the ToF global coordinate system. 85, 86

T_B^W Transformation matrix that aligns the checkerboard plane wrt. the Vicon global coordinate system. 85, 86

P_B^W 3x1 matrix with the 3D position of the set of markers wrt. the Vicon global coordinate system. 85, 86

p_B^W 3x1 matrix with the origin coordinate system of the Vicon global coordinate system. 85, 86

μP_B^W 3x1 matrix with the centroid of the set of markers wrt. the Vicon global coordinate system. 85, 86

B Set of 4 markers in a checkerboard plane. 85

cvM Covariance matrix. 85, 86

V Single Value Decomposition term. 85, 86

U Single Value Decomposition term. 85, 86

dt Single Value Decomposition terms determinant. 85, 86

R_W^C Rotation matrix that aligns the Vicon global coordinate system wrt. the ToF global coordinate system. 85, 86

I 4x4 identity matrix. 85, 86

$(Pxy_B^C)_{t'}$ $2x1xt'$ matrix with the 2D pixel position of the set of markers wrt. the ToF global coordinate system. 86

$(P_B^C)_{t'}$ $3x1xt'$ matrix with the 3D position of the set of markers wrt. the ToF global coordinate system. xiv, 86, 87

$(P_B^W)_{t'}$ $3x1xt'$ matrix with the 3D position of the set of markers wrt. the Vicon global coordinate system. xiv, 86, 87

T_A 4x4 Transformation matrix for the head joint wrt. the MVN Awinda global coordinate system. 87

R_A 3x3 Rotation matrix for the head joint wrt. the MVN Awinda global coordinate system. 87, 88

P_A 3x1 Translation matrix for the head joint wrt. the MVN Awinda global coordinate system. 87, 88

T_J 4x4 Transformation matrix for the each joint wrt. the MVN Awinda global coordinate system. 87, 88

R_J 3x3 Rotation matrix for the each joint wrt. the MVN Awinda global coordinate system. 87

P_J 3x1 Translation matrix for each joint wrt. the MVN Awinda global coordinate system. 87

P_O 3D mesh where each vertex is a rotation of vector P with the Vicon head tracker rotations for each timestamp. 88

P 3x1 vector. 88

R_O^W Rotation matrix that aligns the subject's head joint in the Vicon global coordinate system. 88

R_A^O Rotation matrix that aligns MVN Awinda head joint wrt. the subject's head joint in Vicon system. xiv, 88

P_A^O Translation matrix that aligns MVN Awinda head joint wrt. the subject's head joint in Vicon system. xiv, 88

$RD_{t'}$ Real dataset comprised by the group of dataset information for each synchronized time frame. 89, 116, 164, 166, 168

Pxy_J^C Standard 2D human body pose sample format. MVN Awinda joints 2D pixel position in the ToF image plane. 89, 90, 116, 144, 150, 153, 157, 166, 168

$Pxyz_J^C$ Standard 3D human body pose sample format. MVN Awinda joints 3D position wrt. the ToF global coordinate system. 90, 116, 157, 166, 168

Γ^{NF} Blender user defined number of samples to render. 115

H Number of Blender MakeHuman human models. 105, 116

Af Number of axis of freedom on each joint of a MakeHuman human model. 105, 108, 116

Jf Number of joints that can have freedom of movement on a MakeHuman human model, Γ^T. 105, 108, 116, 121

Γ^T Blender user defined ToF model. xvi, 106, 107, 115

Γ^X Blender user defined ToF horizontal resolution. 106

Γ^Y Blender user defined ToF vertical resolution. 106

Γ^{HFoV} Blender user defined ToF horizontal field of view. 106

Γ^{tp} Blender user defined ToF 3D position. 106

Γ^{to} Blender user defined ToF 3D orientation. 106

Γ^n Blender user defined ToF axial noise model quadratic equation. 106, 118, 119

Γ^a Blender user defined 1^{st} term of from the ToF axial noise model quadratic equation. 106, 110, 118

Γ^b Blender user defined 2^{nd} term of from the ToF axial noise model quadratic equation. 106, 110, 118

Γ^c Blender user defined 3^{rd} term of from the ToF axial noise model quadratic equation. 106, 110, 118

ι Blender internal ToF model created from a user defined ToF model. 106, 115, 121

ι^X Blender internal ToF horizontal resolution. 106, 112, 121

ι^Y Blender internal ToF vertical resolution. 106, 112, 121

ι^{HFoV} Blender internal ToF horizontal field of view. 106, 112, 121

ι^p Blender internal ToF 3D position. 106

ι^o Blender internal ToF 3D orientation. 106

ι^n Blender internal ToF axial noise model quadratic equation. 106

Γ^C Blender user imported/defined car model. xvi, 106, 107, 115

Γ^{cp} Blender user defined car model 3D position. 106

Γ^{co} Blender user defined car model 3D orientation. 106

Γ^{cm} Blender user imported car model mesh. 106

ζ Blender internal car model created from a user imported/defined car model. 106, 115

ζ^p Blender internal car model 3D position. 106

ζ^o Blender internal car model 3D orientation. 106

ζ^m Blender internal car model mesh. 106, 108, 115, 117

Γ^H_h Blender user imported/defined MakeHuman human model. xvi, 105, 107, 115

$\Gamma^g_{h,j,a}$ Blender user defined MakeHuman human model Gaussian motion profile associated to each of its joints axes. 105, 108, 115, 116

$\Gamma^\mu_{h,j,a}$ Blender user defined MakeHuman human mode Gaussian mean value motion profile associated to each of its joints axes. 105, 108, 116

$\Gamma^\sigma_{h,j,a}$ Blender user defined MakeHuman human mode Gaussian standard deviation value motion profile associated to each of its joints axes. 105, 108, 116

$\Gamma^{inc}_{h,j,a}$ Blender user defined MakeHuman human model incremental motion profile associated to each of its joints axes. 105, 108

$\Gamma^{ip}_{h,j,a}$ Blender user defined MakeHuman human model initial pose (3D position and rotation) associated to each of its joints axes. 105, 115

a Blender MakeHuman human model axis indexing. 105

Γ^{rgb}_h Blender user imported MakeHuman human model RGB skin texture. 105, 112, 115

Γ^{lb}_h Blender user imported MakeHuman human model label skin texture. 105, 112, 115

η_h Blender MakeHuman human model. 104, 105

η_h^{sk} Blender MakeHuman human model 3D skeleton mesh. 105, 108, 115, 117

η_h^{sm} Blender MakeHuman human model skin mesh. 104, 105, 108, 115, 117

η_h^{st} Blender MakeHuman human model skin texture. 105, 112

$\eta_{h,j,a}^{bp}$ Blender MakeHuman human model body skeleton with joints and axes. 105, 108, 115, 116, 121

$\eta_{h,j}^{p}$ Blender MakeHuman human model body skeleton joints 3D position. 105

$\eta_{h,j,a}^{o}$ Blender MakeHuman human model body skeleton joints axis orientation. 105, 108, 116

$\eta_{h,j,a}^{rm}$ Blender MakeHuman human model body skeleton joints axis range of motion. 105, 107, 108

np_h Number of possible body poses for each human considering incremental motion profile. 108

h Blender MakeHuman human model indexing. 108, 113

$noisedepth_{x,y}$ Standard format for a single noise based depth frame. Blender rendered ToF noisedepth frame. 110, 111, 115, 119, 120, 166

$noisedepth_{t',x,y}$ Standard dataset noise based depth frame format. Blender rendered ToF noisedepth frame. 116

$\sigma_{x,y}$ Blender standard deviation of the error related to the depth frame associated with the rendering of ToF noisedepth frame. 110

D Number of different distances used for Blender ToF noise model regression. 118

Fr Number of recorded frames in each distance used for Blender ToF noise model regression. 118

$depth_{d,x,y,f}$ Depth frames for each recorded sample for each distance for Blender ToF noise model regression. 118

$\mu_{d,x,y}$ Mean error calculated for each pixel in each distance for Blender ToF noise model regression. 118

σ_d Standard deviation of the error calculated for each distance for Blender ToF noise model regression. 118

$\sigma_{d,x,y}$ Standard deviation of the error calculated for each pixel in each distance for Blender ToF noise model regression. 118

th Blender empirical Gaussian noise threshold associated with the rendering of ToF noisedepth frame. 110, 111

k Blender empirical Gaussian noise kernel x-dimension associated with the rendering of ToF noisedepth frame. 110, 111

w Blender empirical Gaussian noise kernel y-dimension associated with the rendering of ToF noisedepth frame. 110, 111

x_c Blender empirical circular crop x-coordinate associated with the rendering of ToF noisedepth frame. 111

y_c Blender empirical circular crop y-coordinate associated with the rendering of ToF noisedepth frame. 111

ra Blender empirical circular crop radius associated with the rendering of ToF noisedepth frame. 111

dz Blender empirical dead-zone associated with the rendering of ToF noisedepth frame. 111, 119

sat Blender empirical saturation associated with the rendering of ToF noisedepth frame. 111, 119

$nstdepth_{x,y}$ Standard format for a single NST based depth frame. Blender rendered ToF nstdepth frame. 111, 112, 120, 121, 166

$nstdepth_{t',x,y}$ Standard dataset NST based depth frame format. Blender rendered ToF nstdepth frame. 116, 168

x_S Blender NST style frame associated with the rendering of ToF nstdepth frame. xvi, 111, 120

x_C Blender NST content frame associated with the rendering of ToF nstdepth frame. xvi, 111, 119, 120

T_l^r Blender NST guidance channels associated with the rendering of ToF nstdepth frame. 119

T_l^o Blender NST car windows guidance channel associated with the rendering of ToF nstdepth frame. 119, 120

T_l^o Blender NST car interior guidance channel associated with the rendering of ToF nstdepth frame. 119, 120

T_l^o Blender NST human guidance channel associated with the rendering of ToF nstdepth frame. 119, 120

l Blender NST VGG19 layer indexing associated with the rendering of ToF nstdepth frame. 119

r Blender NST spatial regions indexing associated with the rendering of ToF nstdepth frame. 119

$labels_{x,y}$ Standard format for a single labels frame. Blender rendered ToF labels frame. 111, 115, 119, 120, 166

$labels_{t',x,y}$ Standard dataset labels frame format. Blender rendered ToF labels frame. 116

$RGB_{x,y}$ Standard format for a single RGB frame. Blender rendered ToF RGB frame. 112, 115, 166

$RGB_{t',x,y}$ Standard dataset RGB frame format. Blender rendered ToF RGB frame. 116

pcx_f Blender rendered ToF 3D point-cloud matrix x-coordinates. 112

pcy_f Blender rendered ToF 3D point-cloud matrix y-coordinates. 112

pcz_f Blender rendered ToF 3D point-cloud matrix z-coordinates. 112

$(Pxy_J^C)_h$ Blender rendered joints 2D pixel position in the ToF image plane for each human model. 113, 115, 121

$(Pxyz_J^C)_h$ Blender rendered joints 3D position wrt. the ToF global coordinate system for each human model. 113, 115, 121

$SD_{t'}$ Synthetic dataset comprised by the group of dataset information for each rendered sample. 116, 164, 166, 168

u Pixel feature in Shotton's method. 138, 139

nB Yolo number of bounding boxes and confidence score predicted by a grid cell. 141

nB Yolo number of probabilities of condicional class in a grid cell. 141

$\hat{C}_c$ Yolo bounding box confidence score. 141

bb Yolo bounding box indexing. 141

c Yolo cell indexing. 141

t_x Yolo predicted bounding box center x-coordinate. 141

t_y Yolo predicted bounding box center y-coordinate. 141

t_w Yolo predicted bounding box width. 141

t_{he} Yolo predicted bounding box height. 141

$P_r(class_c \mid object)$ Yolo predicted probabilities of conditional class in a grid cell. 141

$(Pxyz_J^C)_{t'}$ Standard dataset 2D body pose format. Human body pose joints 2D pixel position wrt. the ToF image plane. 155, 166

$(Pxyz_J^C)_{t'}$ Standard dataset 3D body pose format. Human body pose joints 3D position wrt. the ToF global coordinate system. 145, 166

Pxy_{head}^C 2D head pose sample. 144

P_{ALG} Algorithm inference of human body pose joints 2D/3D position wrt. the ToF global coordinate system. 145, 166

P_{GT} Dataset human body pose joints 2D/3D position wrt. the ToF global coordinate system. 145, 166

TP Algorithm true positive inference. 145, 146

TN Algorithm true negative inference. 145

FP Algorithm false positive inference. 145, 146

FN Algorithm false negative inference. 145, 146

d_j Joint Euclidian distance between algorithm inference and dataset ground-truth. 145, 146

mAD Joint mean average Euclidian distance between algorithm inference and dataset ground-truth. xxiii, 145, 147, 158, 159, 166

mAD^{10cm} Joint mean average Euclidian distance between algorithm inference and dataset ground-truth for valid joints in a 10cm distance. 145, 147

mAD^{5cm} Joint mean average Euclidian distance between algorithm inference and dataset ground-truth for valid joints in a 5cm distance. 147

mAP^{10cm} Joint mean average precision for valid joints in a 10cm Euclidian distance between algorithm inference and dataset ground-truth. 146

mAR Joint mean average recall for valid joints between algorithm inference and dataset ground-truth. 146, 166

Γ^p Caffe prototxt user input. xix, xxiii, 151, 152, 156, 157, 158, 159

σ_x Caffe prototxt user input for range in scalling in x-axis. xxiii, 157, 158

σ_y Caffe prototxt user input for range in scalling in y-axis. xxiii, 157, 158

σ_z Caffe prototxt user input for range in scalling in z-axis. xxiii, 157, 158

$P_{augment}$ Caffe prototxt user input for probability for scalling. xxiii, 157, 158

$Px_{augment}$ Caffe prototxt user input for probability for scalling in x-axis. xxiii, 157, 158

$Py_{augment}$ Caffe prototxt user input for probability for scalling in y-axis. xxiii, 157, 158

$Pz_{augment}$ Caffe prototxt user input for probability for scalling in z-axis. xxiii, 157, 158

$L2$ PAF heat maps branch. xix, 152

$L1$ PAF part affinity fields branch. xix, 152

$L3$ PAF label detection branch. xix, 152

f_{heat} PAF heat maps loss function. 153

S_j PAF predicted heat map for joint j. 153

S_j^* PAF ground-truth heat map for joint j. 153

P_c PAF predicted part affinity field for connection c. 153

P_c^* PAF ground-truth part affinity field for connection c. 153

pC PAF number of connections betwen the different body joints. 153

F_L PAF label detection branch convolutional layers output feature maps. 154

ci PAF label detection branch output class indexing. 154

K PAF label detection branch output number of classes. 154

$p_{ci,j}$ PAF label detection branch output prediction score with softmax for each class in each joint. 154

$y_{ci,j}$ PAF label detection branch output logit for each class in each joint. 154

$t_{ci,j}$ PAF label detection branch ground-truth score with softmax for each class in each joint. 154

$MD_{t'}$ Manual labelling dataset comprised by the group of dataset information for each sample. 155

Pxy 2D pixel position. 156, 157

$RG_{t',x,y}$ Region growing frame with background removed through a seed position. 144

$CC_{t',x,y}$ Pixel connectivity frame that identifies pixels that are not connected. 144

$frame_{x,y}$ Normalized frame used for encoding dataset frames for algorithmic evaluation. 166

Γ^t User defined threshold for algorithmic precision and distance metric evaluations. 167

Γ^u User defined ToF use case. 166

mAD^{Γ^t} Joint mean average Euclidian distance between algorithm inference and dataset ground-truth for valid joints in a user defined distance. 166

mAP^{Γ^t} Joint mean average precision for valid joints in a user defined Euclidian distance between algorithm inference and dataset ground-truth. 166

$d_{t',j}$ Joint Euclidian distance between algorithm inference and dataset ground-truth for all samples. 167

Chapter 1

Introduction

This chapter presents the concept of human body pose detection, both outside and inside the vehicles, as well as its validation with ground-truth systems. These concepts are the driving factor for the goals and contributions of this thesis. The organization of the dissertation is also listed at the end of this chapter.

With current development and deployment of Advanced Driver-Assistance Systems, higher levels of car automation will be available. With them, the human factor inside the car will also change.

These modifications require advanced systems for passenger detection, their behaviours and the way they interact with the vehicle. Due to this fact, systems that allow for passenger body pose detection inside of vehicles are the determinant factor to ensure both safety and comfort. Currently, there are several known R&D groups already focused in this or in other use cases, such as Laboratory for Intelligence & Safe Automobiles [26, 27, 28, 29, 30], and Stanford Artificial Intelligence - Toyota Center For AI Research [31, 32, 33, 34, 35]. Others show the same focus but outside the automotive industry, such as Fraunhofer Institute of Optronics, System Technologies and Image Exploitation [36, 37, 38, 39], and Carnegie Mellon University (CMU) [40, 41, 42].

To achieve a system able to monitor occupant's body pose, it is necessary to develop a suitable technology for monitoring (i.e. image sensor), and an algorithm to detect the human body pose. This algorithm can be seen as a measuring device, and to validate such device there is the need to do so with a higher performance one (i.e. ground-truth system). This system must be capable of measuring the human body pose, and evaluate the developed algorithm.

Contents

1.1 Ground-truth system

The ground-truth concept is related to the measurements made in cartography, where the remotely captured information from satellites would be validated with measurements made on the ground. In what concerns the information captured by Global Positioning System for investigation in the field of transportation, ground-truth refers to what the traveler is doing (i.e. travel time, distance, etc.) [43]. However, this concept can be applied transversally in other fields of 3D positioning, where to validate/calibrate a measurement device, you need a device that works has a standard (i.e. ground-truth system). This system can and must be contextualized with metrology [44]. Its concept can be applied in its different action fields, namely the scientific metrology, that ensures the organization, development and maintenance of measurement standards; the industrial metrology, that ensures the performance of measurement devices used in industry and industrial processes; and the legal metrology, that has the purpose to ensure the level of accuracy of measurements when they influence the transparency of economical, health and safety transactions. For a better understanding of this system functionality, we can use the applied standard process in industrial metrology, that is represented in Figure 1.1.

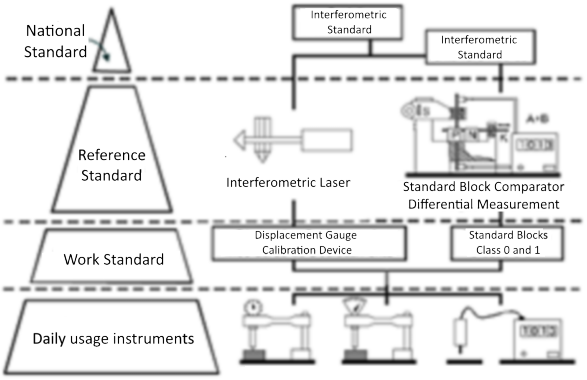


Figure 1.1: Standard process in industrial metrology. Image adapted from [1].

In this process, the main objective is to insure the quality of the measurement devices used by the end consumer. To achieve this, it is necessary to use devices with high accuracy and reduced uncertainty, that will serve as standard for end devices calibration. This system must have fundamental concepts in its conception, namely that its metrological static and dynamic characteristics have a higher performance than the device being calibrated (i.e. usually with a relation of 1:10 or 1:4). As a conclusion, to validate the human body pose estimation algorithm, it is necessary to develop a ground-truth system with 6 Degrees-of-Freedom (DoF) for each human body joint (i.e. motion capture system). As mentioned previously, this system must have a higher performance than the human body pose detection, namely in terms of accuracy, precision,

uncertainty, resolution, range, drift and hysteresis.

In 2006, Moeslund et al. [45] made a survey of advances made between 2000 and 2006 in vision-based human motion capture. With over three hundred related publications identified in major conferences and journals, it was possible to identify the main technological challenges, and the key advances that it:

- **Initialization**: Automatic initialization of model shape, appearance and pose for human motion reconstruction;
- **Tracking**: Tracking of multiple people in unstructured outdoor scenes, using appeareance, shape and motion from figure-ground segmentation;
- **Human motion reconstruction from multiple views**: Techniques to efficiently search for space of possible pose configurations for robust reconstruction from mutiple view video acquisistion. Current approaches capture gross body movement but do not accurately reconstruct fine detail such as hand movements or axial rotations;
- **Monocular human motion reconstruction**: Human motion capture from single views with stochastic sampling techniques using learnt motion models to constrain reconstruction based on movement;
- **Pose estimation in natural scenes**: Probabilistic assemblies of parts based on robust body part detection has achieved 2D pose estimation in challenging cluttered scenes such as film footage;
- **Recognition**: Understanding behavior and action for surveillance applications towards automatic detection of unusual activities.

It is easy to understand that motion capture is of high demand in different areas, leaving us with an increasing number of scientific articles about the subject. From 2006 to 2017, the scientific interest in motion capture stayed relevant. To get a grasp on the current methodologies, it requires a continuous and extensive work. At the moment, more than 400 publications at journals are accounted for.

1.1.1 Motion capture systems

Generating real data for Machine Learning (ML) algorithms is an important task in a wide range of areas. Motion capture data for the in-car scenario can be seen has a difficult task, given the lack of motion capture systems that can reliably work in it. Although there are accurate motion capture systems available, they are not focused in heavily occluded scenarios. The alternative can involve eletromagnetic or inertial based systems, solving the occlusion problem, but adding a new limitation - the magnetic distortion sensitivity.

Eletromagnetic systems

Systems such as Polhemus (Polhemus, Vermont, United States) are eletromagnetic based, and despite being highly accurate, they suffer from eletromagnetic disturbances from external sources, such as metals or electronic devices. Mitobe et al. [46] were able to track finger movement of pianists with a spatial resolution of $3.8\ \mu m$, although it also showed the complexity of the setup and the need for placing a single wired sensor for each tracked joint. Other wireless solutions are available through the Polhemus Liberty latus [47, 48], allowing full body motion capture, but still suffering from the same sources of uncertainty.

Optical systems

Optical based motion capture systems are used across several R&D fields [49, 50, 51] and can be separated in two types: (1) marker-based with high accuracy, illumination imunity and high setup time; and (2) markerless with fast setup time but with reduced accuracy and higher sensitivity to light conditions. Although marker-based systems, such as the Vicon system (Vicon, Oxford, UK), are the gold standard for motion capture, they suffer from the fact that they need to have the markers in line of sight to guarantee acurate tracking. Rahmatalla et a. [52] circumvented the occlusion problem by adding virtual (calculated) markers while tracking a seated operator in-lab. However, they only focused on the lateral pelvis' markers. Considering the technical limitations of the most robust motion capture systems, it is not feasable to use them alone for the in-car environment.

Inertial and magnetic measurement units (IMMUs)

Inertial and Magnetic Measurement Units (IMMUs) based systems are an alternative to optical systems since they do not need the subject to be in line of sight. Theoretically, they are able to infer body segment orientation as well as joints' positions, althought they are prone to errors caused by drift or magnetic sensitivity. Another issue for the estimation of full body kinematics is related to the need of a biomechanical model and its initial calibration. Current biomechanical models are proposed by the Internal Society of Biomechanics (ISB) [53]. For example, Xsens (Xsense, Enschede, Netherlands) uses a modified version with 23 segments and 22 joints [54]. During calibration, the subject needs to stand in a calibration posture, known as the N-Pose or T-Pose, which assumes that all segments are aligned and the coordinate systems of all joints are parallel to one another. This initial assumption adds a systematic error that offsets the segments' orientations and joints' positions. In its thesis [55], Monica Orozco compared MVN BIOMECH Awinda from Xsense against Vicon Nexus through the study of gait kinematics with calibration postures that deviate from the standard

N-Pose, showing that the error introduced could be considered as a shift in joint angle values while the shape was not affected. It was also possible to understand that this error could be corrected with the information of the true body posture captured during the calibration procedure. Two correction approaches were proposed (Orientation Correction and Planar Angle Correction), giving the possibility to achieve an initial calibration procedure for human subjects that are not able to attain a N-Pose.

1.2 Human body pose detection

Human body pose detection represents an extensive research challenge for a great deal of applications. In general, these methods can be divided in two main classes: generative and discriminative. Generative approaches are designed to fit and deform a model to match with the image to detect the pose [5, 56, 57, 58, 6, 59, 60, 7, 61]. Discriminative approaches are designed to learn a mapping from image features to a body pose, using only the information of the image [2, 22, 3, 4, 62, 63, 39]. To define the best approach to detect the human body, it is needed to weight the pros and cons of both methodologies (Table 1.1).

Table 1.1: Algorithmic classes.

Classes	**Advantages**	**Disadvantages**
Discriminative	-No need for tracking method -Fast after training -Without error accumulation -Efficiency	-No explicit model -Big training dataset
Generative	-Explicit model -Correlation between body parts	-Needs initialization -Slow after training -Local minima

Regarding the generative ones, the main advantage is the robustness, once these methods fit one or more previous models to the image, allowing to introduce shape prior information to the method. However, this class of methods uses error minimization functions, being susceptible to be trapped in local minima and having a high computational cost. In opposite, the discriminative methods are fast after training, allowing to achieve real time detection. Moreover, these methods are capable of dealing with large body shape variations. The main problem associated with discriminative approaches is that they are inherently limited by the amount and the quality of the training data, requiring a large training dataset to avoid overfitting and generalization problems. Weighting the advantages/disadvantages of each methodology and considering

that a pose detection algorithm with real time capability is needed in our scenario, it was decided to follow a discriminative approach.

1.2.1 Discriminative Methods

Depth based discriminative approaches consist on different methods. Body part detection of the human body can be done by identifying salient points [64], or regression methods by regressing a mapping function from a large set of training data [2, 22, 3, 4, 63, 39]. This approach has the potential of generalizing to different body shapes, invariant to pose, body shape, clothing. The processing efficiency is high, allowing higher frame rate with enough DoF information related to the skeletal tracking. The down side is that the training process is computationally expensive (i.e. Kinect API requires 300k poses to be trained in 24 hours on a 1000 core cluster). Although the inference is fast and reliable, it does not work well with occlusion, because the processing is done frame-by-frame and it does not take into account the movement in time and its prediction. To develop the present Kinect Application Programming Interface (API) for skeletal tracking, Shotton et al. [2] used a large and varied (real and synthetic body pose data) training dataset that allowed the classifier to estimate body parts invariant to pose, body shape, clothing, etc. The 900k poses were filtered to 300k by just constraining the minimal distance between joints ($5\ mm$). They were used to train a Random Forest Classifier (RFoC) for body part segmentation from a single depth image. The system runs at 200 frames per second (fps) on a consumer hardware, where the computational processing was being done on the Xbox Graphics Processing Unit (GPU) (Figure 1.2).

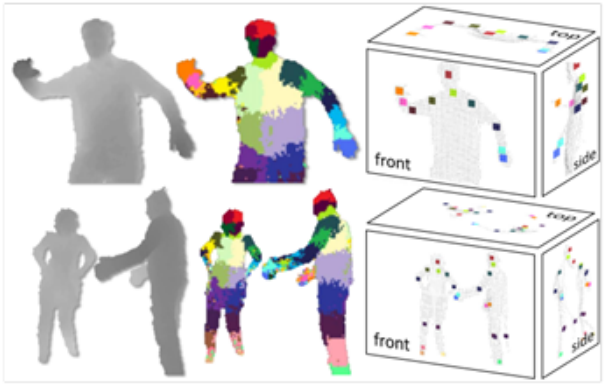


Figure 1.2: Shotton et al. human body pose detection algorithm: from depth, to body parts, to 3D joint proposal. Image adapted from [2].

Buys et al. [3] used a similar approach to Shotton et al. [2] with some differences residing on the inferring process. Shotton et al. assumed that the camera had a fixed position, so it was possible to first do a depth based background subtraction and then evaluate the pixels with the Random Decision Forest. Buys et al.

assumed that the camera did not have a fixed position so it had to evaluate all pixels without doing depth based background subtraction, which reduces the accuracy of the skeletal tracking. To solve this, there was a second stage after pose inferring (i.e. body part segmentation inferring followed by joint position calculation), where the RGB body silhouette was used to remove wrong pixels, so a second body pose inferring could be done and enhance the accuracy from the body pose detection (Figure 1.3).

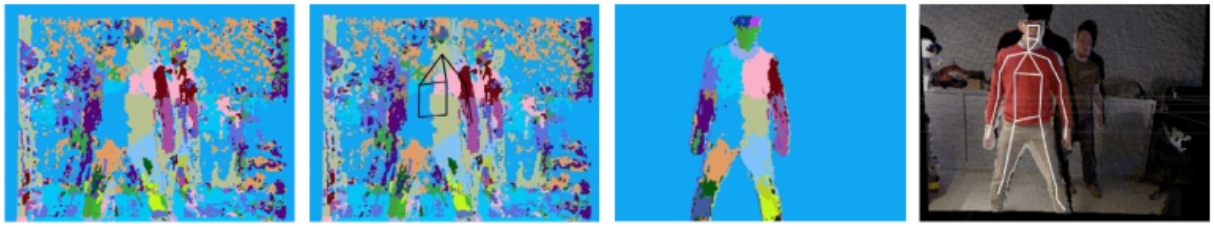


Figure 1.3: Buys et al. two stage RGB-D human body pose estimation. Image adapted from [3].

Estimating upper body poses from sequence of depth images is a challenging problem. Because of the adoption of randomized forest methods to label human parts in real time, it requires enormous training data to obtain favorable results. Tsai et al. [4] proposed a novel two-stage method to estimate the probability maps of upper body parts of users (Figure 1.4). These maps are then used to evaluate the region fitness of body parts for pose recovery. The probability maps estimated by the classifiers are applied to objective functions for skeleton extraction, where a Random Sample Consensus method is used to estimate approximate results. The framework was adapted to parallel computing for GPU processing and real time detection. The two-stage method was compared with a single-stage RFoC for upper body estimation. For the single-stage RFoC method, 30000 synthesized depth images were adopted, of which the postures were acquired from the CMU mocap database. For the two-stage estimation model, 10000 synthesized images to train the first upper and lower body classifier. For the second phase, 20000 images were adopted to train the detailed upper-body-part classifier. In the end, the two-stage method achieved more stable results with compact training data and real time capabilities.

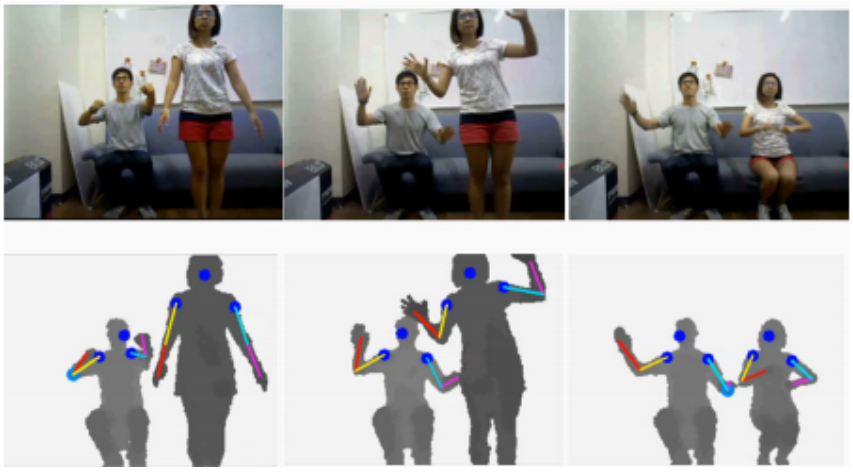


Figure 1.4: Tsai et al. human upper body pose estimation. Image adapted from [4].

Hesse et al. [39] proposed a system with fast training and flexibility to fit the requirements of markerless 3D movement analysis of infants. It uses Random Ferns Classifier as a robust alternative to RFoC to find the 3D positions of body joints in single depth images. The training time is reduced by several orders of magnitude compared to the Kinect approach using a similar amount of data. The system was trained in 9 hours on a 32 core workstation opposed to 24 hours on a 1000 core cluster, achieving comparable accuracy to the Kinect Software Development Kit (SDK). The average distance error over all joints was $41\ mm$.

Crabbe et al. [63] proposed a method for estimating body pose in high-level pose spaces directly from a depth image without relying on intermediate skeleton-based steps. The method is based on a Convolutional Neural Network (CNN) that maps the depth-silhouette of a person to its position in the pose space. The pose representation is initially built from skeleton data. This opens the possibility for a wider application of the movement analysis method to movement types and view-angles that are not supported by the skeleton tracking algorithm. Training took 7 hours and algorithm performance was of 100 fps.

Discriminative strategies rely on two main phases: 1) training phase where the method is trained using a dataset (e.g. depth images) with the respective labels (e.g. 3D joint position); and 2) testing phase where the trained method is used to predict unlabelled samples (Figure 1.5A). The training phase normally starts with feature extraction from the images. A feature extractor strategy is used to transform an image (i.e. pixel intensities) into other type of representation (e.g. Histogram of Oriented Gradients (HoG), Scale-Invariant Feature Transform) that helps the discriminative method to classify or detect patterns on the image. After that, these features and the corresponding labels are used to train the method. In the testing phase, the same feature extraction strategy is applied, which will be used by the trained method to predict the label. Inside the discriminative class, we can define two sub-classes: the traditional and the Deep Learning (DL) methods (Figure 1.5B). The majority of methods implemented so far to human pose estimation were based

in traditional machine learning approaches (e.g. Support Vector Machine (SVM), RFoC, Neural Network). However, in the last years, human detection and pose estimation methods based on deep learning approaches appear forcefully. The major difference between these two types of discriminative methods lies in the fact that the traditional ones need hand-crafted feature extractors. In a DL approach, this step is eliminated as the method selects the best features by itself. Since the features extraction step requires a lot of time and its design depends directly of the expertise of the user, deep learning approaches open the possibility to achieve more accurate results than traditional machine learning strategies.

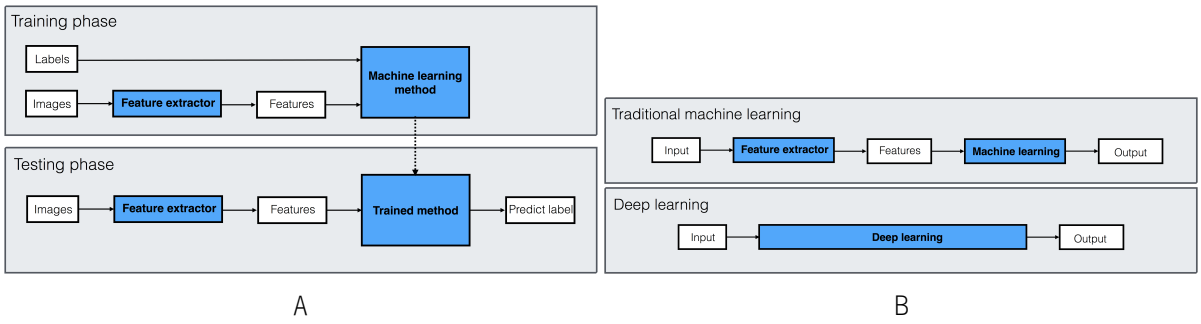


Figure 1.5: (A) Discriminative method phases: in the first phase, the method trains in a labelled dataset; in the testing phase, the trained method is used to classify unlabeled images. (B) Comparison between traditional machine learning and deep learning approaches.

1.2.2 Generative Methods

One difference between discriminative and generative approaches is the time consideration. Where the first is done frame by frame or multi-frame (i.e. through time with Recurrent Neural Networks or Long Short-Term Memory). The second takes time into consideration for human body pose estimation, while using prior knowledge on human kinematics, body shape and/or temporal motion continuities. Among other things, these priors allow to create a system without the need of a large body pose training dataset. Unfortunately, this leads to problems related to tracking, both on the automatic initialization process and recovery from loss of tracking. This approach can resort to different types of filtering [6, 60], Kalman Filter (KF), Extended Kalman Filter, Particle Filter (PF), Partitioned Particle Filter (PPF), Annealed Particle Filter, Condensation Particle Filter, Particle Swarm Optimization (PSO). The filtering objective is to predict joint positions while taking into account previous state, future prediction state and measured state. This way it is possible to better estimate the possible joint position and orientation.

There are also other generative approaches where a template, either parametric or non-parametric, is fitted to the observed data, mostly with variants of Finite Iterative Closest Point (FICP) [5, 56, 59, 7]. This method uses a source point cloud (captured data) and a reference point cloud (template) so it can estimate the

combination of rotation and translation that will best align both. The template adaptation poses a consistency problem between template and subject's body shapes. One option is to use pre-scanned personalized models, where the scans can occur prior to the body pose detection, with one or multiple depth sensors. Another option is to adapt the template either using dedicated frames or along the tracking process. Knoop et al. [5] proposed a tracking system for 3D tracking of human body movements based on a 3D body model and FICP. The proposed approach was able to incorporate raw data from different input sensors, as well as results from feature trackers in 2D or 3D (Figure 1.6).

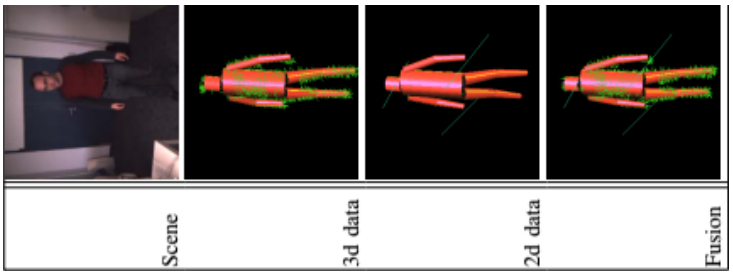


Figure 1.6: Knoop et al. 2D and 3D human body pose detection fusion. Image adapted from [5].

Pekelny et al. [56] presented one of the first approaches for pose and surface estimation from Time-of-Flight (ToF) Depth frames. An initial segmentation on the first frame is done, including the knowledge of the kinematic structure. FICP is used to continuously align the model from the first frame with subsequent depth frames. Xing et al. [6] modeled the human body as an assembly of 3D geometric primitives whose dimensions were estimated automatically (Figure 1.7). Motion parameters were recovered by projecting hypothesized body model pose to camera imaging space and seeking for optimal solution that best matches camera observation as well as physical constraints. The temporal approach was done by designing an objective function to quantify the discrepancy between the predicted and the actual observed features and penalize implausible or unnatural poses. These authors exploited the body skeleton's tree structure and proposed a self-adaptive version of PSO to solve the optimization problem. In the end, it was possible to achieve higher accuracy then the Kinect skeleton tracking, running at 30Hz. Nonetheless, the test sampling data was not every extensive, impeding a better grasp on its true accuracy.

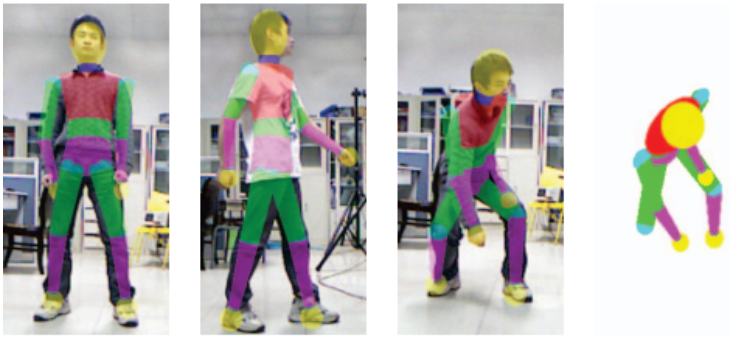


Figure 1.7: Xing et al. human body pose model: (A, B and C) three examples of different poses with the estimated 3D body model superimposed on the color images, (D) model in pose (C) from a different Point-of-View (PoV). Image adapted from [6].

Ganapathi et al. [59] derived an algorithm for tracking human pose in real time from depth images based on Maximum a Posteriori Estimation in a probabilistic temporal model. Their key idea was to extend the FICP by modeling the constraint that the observed subject could not enter free space. The resulting filter ran at 125 Hz on one Central Processing Unit (CPU) core.

In its thesis [60], Li developed a novel full body pose tracking system which incorporates a previous body part detection algorithm into a particle filter framework. The Local Shape Context (LSC) feature was generalized to not only recognize the endpoint body parts, but also to detect the limbs simultaneously. The LSC descriptor was used to describe the local shape of different body parts with respect to a given reference point on the human silhouette, and was shown to be effective at detecting and classifying endpoint body parts, while being computationally efficient. A successful tracking was achieved by using a PF with partitioned sampling. Particle likelihood was evaluated using the identified endpoint body part locations, the depth map, and other detected body part information. The full tracking space (25 DoF) was partitioned into 9 partitions. The particles first were propagated, evaluated/weighted, resampled in each of the partitions, and then each particle is evaluated one more time by considering all the partitions to estimate the final distribution. The particle weighting used several cues, namely explicit distance cue, depth cue and implicit distance cue. The system is capable of tracking in near real time on a standard laptop computer, achieving an average error of 10 cm.

Ye et al. [7] presented a novel real time algorithm for simultaneous pose and shape estimation for articulated objects, such as humans and animals (Figure 1.8A). The pose estimation component embedded the articulated deformation model with exponential-maps-based parametrization into a Gaussian Mixture Model. Benefiting from this probabilistic measurement model, the algorithm requires no explicit point correspondences as opposed to most existing methods (for example FICP methods). In regard to the template used,

its novel shape adaptation algorithm based on the same probabilistic shape model automatically captures the shape of the subjects during the dynamic pose estimation process, improving the tracking accuracy. In addition, two novel approaches were used, a mesh model or a sphere-set model as the template for both pose and shape estimation. The algorithm is fast and accurate, achieving $60\ Hz$ on a mid-range GPU. Unfortunately, the algorithm still presents problems when body members are close to other body parts, and when non-rigid surfaces create body occlusion, (e.g. skirts; Figure 1.8B). Nonetheless, the algorithm achieved a higher accuracy than other concurrent approaches, both discriminative [2] and generative [65, 59].

A B

Figure 1.8: Ye et al.: (A) pose and shape estimation for human and animals; (B) failed body pose estimation. Images adapted from [7].

1.2.3 Hibrid Methods

In the interest of achieving the higher accuracy of generative approaches and the higher frame rate (lower computational power) of the discriminative ones, some work has been done to combine the complementary characteristics of both approaches [65, 8, 29, 38, 66]. Ganapathi et al. [65] combined an accurate generative model with a discriminative model that provides data-driven evidence about body part locations. In each filter iteration, a form of local model based search is applied, that exploits the nature of the kinematic chain. As fast movements and occlusions can disrupt the local search, a set of discriminative trained patch classifiers are used to detect body parts. Unfortunately, the system had a processing time of $[100; 250]\ ms$ per frame to estimate the joint angles. Baak et al. [8] rely on a dataset to infer the pose and later on refines it (Figure 1.9). In this case, salient points features are extracted to be used for dataset lookup. With a variant of Dijkstra's algorithm, the first five geodesic extrema are stacked as the index that is used to infer the most similar pose in the dataset. The late-fusion scheme is based on an efficiently computable sparse Hausdorff distance to combine local and global pose estimates.

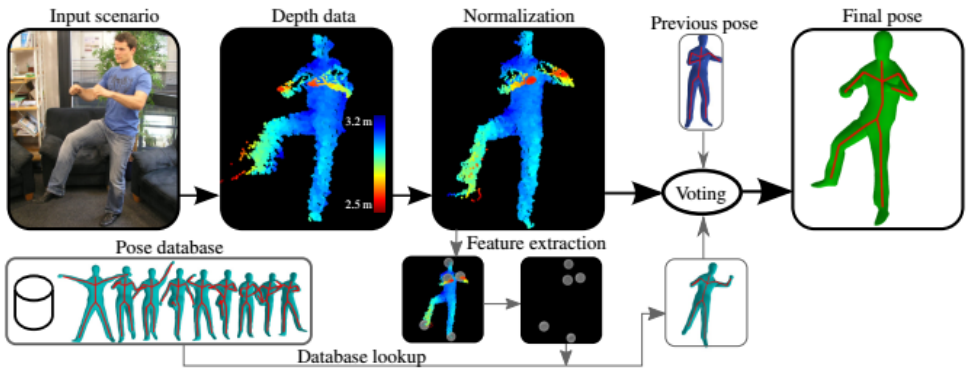


Figure 1.9: Baak et al. human body pose detection framework. Image adapted from [8].

In the interest of developing a method for human-machine interface application in car, Ohn-Bar et al. [29] developed a vision-based system with RGB-D capture, composed by a combined RGB and depth descriptor. A careful evaluation of different temporal descriptors showed the challenging nature of the dataset, with RGB-D fusion proving to be beneficial for recognition. Muench et al. [38] developed a method for action recognition based on the Generalized Hough Transform. The advantages of the proposed method are inherent time segmentation, view-independence, and robustness against noise. The method was validated on a standard hardware using the Kinect sensor for acquiring 3D poses of persons, and it had a processing time of $30\ Hz$. The method acquired skeletal tracking (body pose) as input data, and encoded its information. The encoded data was neither euler angles or quaternions but instead focused on direct cosine data, where the relative joint angle was used. This information was later used to create Hough "templates" to identify body actions. In its thesis, Ville Stohne [66] developed a real time filtering for the human pose estimation using multiple Kinects. To improve the skeletal tracking done by the Kinect API that suffers from occlusion of body parts and limitations regarding rotation, the author proposed a working approach to combine data from multiple Kinect depth sensors to create stable pose estimates of human user in real time. The skeletal tracking (body pose detection) was not part of the development, and the Kinect API (discriminative) was used to gather this information in real time. Stohne focused on the skeletal data fusion from the multiple Kinects, comparing two different filtering techniques, PF and KF. The implemented system provides a stable, fast and cost-efficient setup for motion capture and pose estimations of human users, where the comparison difference between KF and PF was that the first had more performance with good accuracy, and the second increased the accuracy at the cost of performance.

1.3 Datasets

Due to the high demand in the field of investigation for motion capture systems, and to avoid redundancies, several datasets of motion capture data have been made available. These datasets can have a real or synthetic basis, however both follow the same basilar principal - i.e. one must record ground-truth label information wrt. the feature sensor frame (e.g. RGB, ToF, etc). These datasets give access to high amounts of generic information that allows to train algorithms to achieve high accuracy with increased generalization.

1.3.1 Real datasets

The CMU Graphics Lab Motion Capture Database [51] is by far one of the most extensive dataset of publicly available motion capture data. The dataset is comprised by human body poses that include markers' 3D/2D positions and human skeleton data, and RGB frames. All information is directly obtained from the Vicon system, both for the Vicon skeleton template and RGB Vicon cameras. Currently, there are 2605 trials in 6 categories and 23 subcategories, but unfortunately none of them are related to in-car scenarios. The HumanEva Dataset [49] is another database for human motion and pose estimation. The toolchain is comprised of a method of synchronized recording of multiple RGB video sources and 3D/2D motion capture data. The similarities between CMU and HumanEva are related to the human body pose that derives from the Vicon system. Both systems give an accurate body pose, but unfortunately such method cannot be used for in-car scenarios, where occlusion is a factor that invalidates the Vicon system. Within the in-car scenario, Borghi et al. [67] used the Pandora dataset for the POSEidon head and shoulder pose estimator. The Pandora dataset is generated in a laboratory environment with minimal occlusion, with different subjects performing similar driving behaviours while seating on a chair. Head and shoulder orientation were captured through inertial sensors.

Other datasets, such as the SPHERE-Staircase [63], expand the image format to RGB-D. Simpler RGB datasets with 2D ground-truth are also available, such as Vision for Intelligent Vehicles and Applications-Hands/Faces [68], MPII human pose dataset, Leeds Sports Pose, FLIC dataset [69], and Ultrecht Multi-Person Motion Dataset [37]. Even though there are several motion capture datasets to be used as ground-truth and benchmarking tools, to our knowledge there are not many that use RGB-D capture and contain ground-truth data (with higher accuracy than the body pose algorithms). There are simpler datasets that record ground-truth data through algorithmic inferring and not as real ground-truth data.

Due to this limitations, it is of interest to this thesis, the creation of a real in-car motion capture dataset containing synched information from the image sensor and human body pose information from the ground-

truth system. This dataset can be used for the algorithmic evaluation, and if it is made available it can be used as a community tool for the same purpose.

1.3.2 Synthetic datasets

Another approach is to build a synthetic dataset, through the support of other real motion capture datasets. While CMU has an extensive motion capture database, its data consists of ground-truth data and RGB frames only. Its ground-truth data can be used separately for computer graphics animation, as well as to build a virtually animated system that simulates depth frames from a ToF camera, taking into account the sensor Field-of-View (FoV), noise mode, etc. In this type of scenario, it is possible to animate a human body mesh with the ground-truth data from CMU, add the camera perspective, noise model, and export depth frames with associated ground-truth information. Shotton et al. [50] created a realistic synthetic dataset from generated depth images of humans of many shapes and sizes in highly varied poses, sampled from a motion capture database. Interestingly, this dataset was not used for the evaluation of the algorithm, instead it was used for its training (i.e. evaluation was done with real data).

For scenarios where it is hard to generate robust datasets, such as in-car, or even when a larger quantity of data is required, synthetic data is an alternative. Varol et al. [70] created the SURREAL dataset and toolchain. The toolchain can not be seen as a fully synthetic dataset generation pipeline because it relies on real motion capture data. Using the Blender engine (Blender Foundation, Amsterdam, Netherlands), real motion capture data from CMU and Human3.6M [71] datasets is fitted into a Skinned Multi-Person Linear Model (SMPL) [72], which is textured and clothed, while the scenery is comprised of a background image, light and a camera that renders depth, RGB, surface normals, optical flow (motion blur) and segmentation frames, as well as ground-truth data for body joint locations (2D/3D).

Outside the in-car human body pose focus, the SYNTHIA dataset was also created. Ros et al. [73] proposed a virtual world to automatically generate realistic synthetic images with pixel-level annotations, something that would be extremely time consuming if it had to be done manually. The entire urban scenery is customizable through the Unity engine (Unity Technologies, San Francisco, United States), including urban object placement, textures, weather seasons, time of day and clouds with dynamic illumination engines. Two datasets are generated: 1) the SYNTHIA-Rand that consists in RGB and labeled frames from multiple cameras placed in the city and 2) SYNTHIA-Seqs that simulates multiple depth cameras on top of a moving virtual car, generating 360° LIDAR data. The toolchain does not focus in RGB sensor realism, but instead focuses in the generalization of the scenery variables, and the automated generation of sensor image and segmentation data for training and evaluation. For object detection, the VANDAL dataset was developed [74]. A semi-automatic

procedure was used to gather 3D CAD models from Web resources, allowing to generate depth images for each one. Object classes were manually queried and downloaded from 3D CAD model repositories, making a total of 319 categories with 30 objects each. Through the Blender engine and a Python script, depth data was generated for each object, while increasing the dataset size by changing object orientation with respect to the camera and its morphology. The last two toolchains [72, 74] focus on the advantages of 3D engines, such as Unity and Blender, enabling the access to easily customizable scenes and generation of new data from it. Also outside the in-car scenario, there are more recent approaches that make use of the Pandora dataset to exploit the generation of synthetic depth images through a GAN approach [75] to tackle the head pose estimation problem. For the same estimation problem, other methods are able to train only on synthetic images [76], however the feature input is based on RGB images, making it a more reach feature frame but less robust to light conditions.

Considering that synthetic visual data realism is important and relevant for algorithmic accuracy, researchers have made efforts to better understand the existing sources of noise in ToF sensors, and to recreate them synthetically. Planche et al. [77] identified and simulated several noise sources, such as axial and lateral, specular surface, non-specular surface, structural, lens distortion, quantization step, motion, rolling shutter and shadow noise. These authors were also able to show how important it is to add realism to the frames when training a CNN, by comparing against different types of synthetic datasets. Some researchers have also avoided the noise source's recreation into frame rendering, and instead focused in the final image noise characterization. BlenSor did this in a simplified way [78]. It focused on generating similar data for various range scanners, while synthesizing Gaussian noise and lens distortions. The procedure of generating depth data does not involve using the depth information of each pixel in Blender, instead performing ray-casting for each pixel, in some way simulating the ToF physical model. Considering a less generic noise model, Nguyen et al. [79] focused on axial and lateral noise regression for the Kinect ToF output frames. Later on, Iversen et al. [80] were able to improve this model and synthetically reproduce the cameras axial and lateral noise also.

1.4 Motivation

Development of a human body pose detection system for the in-car scenario requires two basilar systems: (1) a standardized robust image sensor for real time feature aquisition, and (2) a human body pose detection algorithm with real time performance, high inference accuracy and generalization. In regard to the 2^{nd} system, traditional discriminative based algorithmic development requires datasets with proper ground-truth

data, both for training and evaluation (i.e. in supervised learning). This approach becomes even harder to acomplish in the in-car scenario, due to the scarse or non-existent datasets depending on the use-case, as well as the non-existent ground-truth systems. Therefore, it is of high importance to develop/implement such ground-truth system and image sensor, and consecutively generate these datasets. With proper datasets available, one may then develop the final human body pose detection algorithm for the in-car scenario.

1.5 Goals

This thesis focuses on the development of a pipeline to obtain a human body pose detection algorithm for the in-car scenario. The primary focus will be placed in the development of toolchains that are able to generate datasets for the algorithmic training and evaluation, as well as the evaluation of the ground-truth system. In addition, upon evaluation of state-of-the-art algorithms, an adapted human body pose detection algorithm will be implemented, with post-customization to the scenario in question. The development follows the pipeline illustrated in Figure 1.10.

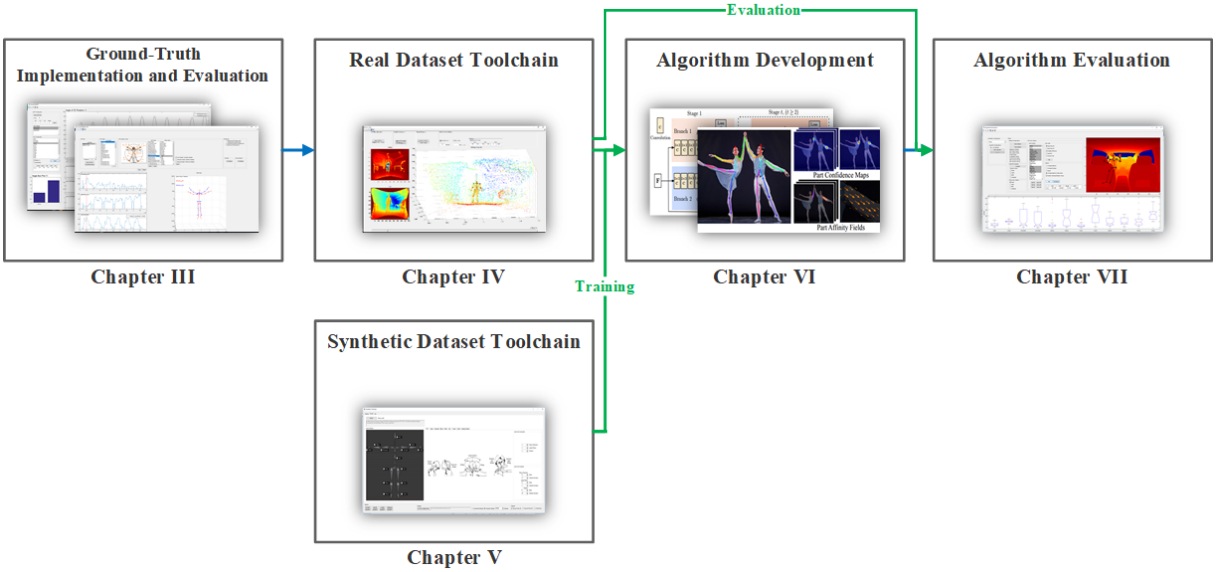


Figure 1.10: Overview of development pipeline.

1.5.1 Ground-Truth Implementation and Evaluation

One of the most important requirements of the real dataset generation toolchain is the availability of a robust human body pose ground-truth system capable of achieving human motion capture in highly occluded scenarios such as in-car. This type of system is not available as a whole, nor as a robust human joint cartesian position capturing system. This task focuses in doing a state-of-art overview of such systems, as well as doing

a market selection of specific products capable of creating a ground-truth system. Before using such systems for the real dataset generation, one must evaluate the proposed ground-truth system. For the evaluation step, several evaluation methods and corresponding toolchains were implemented, capable of recording and evaluating the ground-truth system. This step requires software able to record in real time with proper synchronization between systems, as well as easy data manipulation and evaluation pipelines.

1.5.2 Real Dataset Toolchain

Real datasets are the primary choice to be used to train every detector. This decision comes from the fact that a real dataset is recorded with real sensors feature data, which will later be used to infer the pose. This data not only gives the required information for pose inferring, but also gives information about the sensor's inherent noise model, allowing to generate a training dataset with similar characteristics to the one where the human pose must be detected. Usually, creation of a real dataset consists in recording the body pose with a motion capture system, while synchronously acquiring an image using an image sensor (e.g. RGB or ToF). The pose is then referenced from one system to the other, in order to be correctly projected into the visual frame. This entire procedure is very time consuming and requires a lot of manual interaction, hampering the task of creating a large and generic dataset. Despite some datasets related with human body pose being publicly available (with RGB and depth sensors), these motion capture datasets do not focus on the in-car scenario, mostly due to the fact that standard vision based motion capture systems are not able to function properly in this scenario. This thesis addresses this need by presenting a user-friendly toolchain to generate human body pose datasets in an in-car environment, with the ground-truth system itself being also evaluated. These datasets may be used to train human body pose detection algorithms for an in-car scenario. This toolchain uses an inertial suit for human body pose capture, a ToF sensor for image capture and the Vicon to spatially and temporally align all systems. The output data is comprised of human body poses given with respect to the camera's coordinate system in 2D and 3D for the ToF's amplitude/depth image frame and tridimensional point cloud, respectively. The entire real dataset generation procedure requires an in-lab car scenario, something possible through a car testbed equiped with the real dataset hardware and software technologies. This toolchain requires the same software philosophy as the previous one (Section 1.5.1) due to the complexity of the entire system. The datasets recorded will be used for the algorithm training and evaluation, and can be seen as the best ones for it due to its inherited realism.

1.5.3 Synthetic Dataset Toolchain

Alternatively, synthetic datasets can be used to solve some of the real datasets problems. Unlike real datasets, synthetic datasets can be generated in large quantities and with higher generalization, allowing to obtain more training data in less time. However, synthetic images may suffer from the lack of realism when it comes to modeling the visual sensor noise. Creation of synthetic datasets can be performed using two types of approaches. On the one hand, the procedure can be entirely synthetic and automatic, where the pipeline creates the input and output models [73]. On the other hand, some procedures can still include real data as an input for the pipeline, using recorded motion capture data to give motion to a synthetic generic human model [2, 50]. This thesis addresses the synthetic dataset generation by presenting an automated and user-friendly customizable toolchain to generate realistic human body pose synthetic datasets in a in-car environment. This data must simulate a real in-car environment, allowing to maintain the robustness of the pose detector method when applied to real images. This toolchain uses humans, camera and car models as inputs, where all of them can be customized according to the data to be generated. The output data is comprised of human body poses referenced to the camera's axes, as well as the corresponding image frames from RGB and ToF cameras. For every new frame that is generated, a new per-joint motion is applied to each human model, and it gets validated for collisions between all models. Although both ToF and RGB frames are generated, ToF is the one with more relevance, because of its ability to provide 3D information of the scene, thus giving the detector an easier way to infer an accurate 3D body pose. The entire toolchain follows a strictly software based implementation, capable of creating a car scenario similar to the previous toolchain (Section 1.5.2) but this time more generic and open to customizations, that would not be possible otherwise. While this toolchain uses comercially available software to ease 3D scene manipulation, it implements on top of it an highly customized and automated script. The script implementation allows the toolchain to be oriented with the development needs of the pipeline. The generated datasets will be used for the algorithm training, and although not as realistic as the real ones, they can improve the generalization of the algorithm.

1.5.4 Algorithm Development

To detect the human body pose in images acquired with camera sensors and, thus, the activity of the car's occupants, a robust method for human body pose estimation is needed. Specifically for the in-car environment, few strategies were proposed. Indeed, the images acquired inside a car produce several occlusions, impeding the detection of some joints. This problem hampers the direct usage of traditional human pose estimation algorithms and makes this task more challenging than open space pose estimation.

This thesis addresses this need by focusing on the development of a method for human body pose estimation with a ToF sensor inside a car. Taking into account the timeline of dataset avaiability, a first evaluation to pre-selected methods was performed with a publicly available dataset both for training as well as infering. Although this dataset does not represent the in-car scenario, it does use similar data formats for the image sensor as well as the ground-truth. Through a first algorithmic evaluation/selection step, the next step consisted in customizing the selected method, Part Affinity Fields (PAF) [25], to better suit our generated datasets format. A final step involved the training and evaluation through generated datasets with manually labeled ground-truth. The main requirements of the method were its accuracy in detecting the pose along with its robustness to deal with large variability between different people (i.e. regarding body shape or size). Moreover, to deal with fast human movements, it was also important to guarantee the real time capability of the pose detection.

1.5.5 Algorithm Evaluation

A toolchain to quantitatively and qualitatively evaluate the developed algorithm was then developed, with several requirements in mind:

- ToF camera interface;
- Algorithm interface through language abstraction interface;
- Dataset interface through preestablished dataset standard (real and synthetic);
- Quantitative evaluation through standard metrics;
- Qualitative evaluation through frame projection of dataset ground-truth and algorihm infering.

This toolchain helps mostly on the qualitative evaluation procedure across the entire dataset.

Finally, the algorithm from Section 1.5.5 was evaluated through several training configurations of generated datasets, from real datasets (Section 1.5.2) to synthetic ones (Section 1.5.3). Qualitative and quantitative evaluation was performed through the algorithmic evaluation toolchain (Section 1.5.5). One of the biggest considerations for algorithmic training was the use of a computationaly capable hardware system, which was used transversally across the entire development pipeline.

1.6 Main Contributions

The main contributions of this thesis are two novel toolchains for dataset generation: (1) a real dataset generation toolchain, and (2) a synthetic dataset generation toolchain. In regard to the 1^{st} toolchain, its

inertial-based ground-truth system can be seen as a novelty for the in-car scenario, together with the thorough evaluation procedure presented. In regard to the 2^{nd} toolchain, its major contributions are:

1. Realistic synthetic human body pose generation;
2. Ability to customize car, image sensor, car occupants and their motion profile;
3. Rendering realism is achieved by image processing techniques to mimic camera's output;
4. High throughput of synthetic data.

Both toolchains were developed with the focus to train and evaluate a human body pose detection algorithm, and are currently being used by Bosch. Four datasets were made publicly available to the R&D community, being 2 real and 2 synthetic ones. This thesis does not propose a novel algorithm, however it presents a customized DL based algorithm for the in-car scenario:

1. Detection of the existence of a joint label in the image;
2. Data augmentation strategy for depth images.

The algorithm was trained and evaluated with several datasets generated with the developed toolchains, giving a good understanding of the requirements for improving algorithm training and ultimately accuracy.

1.7 Structure of the Thesis

Chapter 2 describes the hardware and software tools used across all chapters.

Chapter 3 describes the extensive evaluation done on the selected ground-truth system. The overal evaluation methodology is presented. Static evaluations are performed at a sensor level, followed by dynamic full body evaluations at a sensor and joint level. The entire evaluation procedure allows for a better understanding of the error propagation in the system. This method was first presented in **"A system for the generation of in-car human body pose datasets"**, currently in revision process.

Chapter 4 describes the first toolchain for dataset generation, where the development methodology of the real dataset toolchain is presented. The related work on real dataset generation is summarized. The methods for spatial and temporal calibration of all subsystem's of the toolchain are presented. The system's full alignment for dataset generation is shown. The system's evaluation and the potential interest is presented and discussed, as well as the main conclusions for it. This toolchain was first presented in **"A system for the generation of in-car human body pose datasets"**, currently in revision process.

Chapter 5 describes the second toolchain for dataset generation, where the development methodology of the synthetic dataset toolchain is presented. The methods for scene customization, generation and validation are presented. Human model creation and motion profile considerations are also shown. The rendering pipeline tries to achieve realism through an implementation of noise sources and stylization. The toolchain's computational performance and potential interest is presented and discussed, as well as the main conclusions for it. This toolchain was first presented in [81].

Chapter 6 presents the human body pose detection algorithm development. The selection and evaluation of ML and DL based algorithms is shown. The customization of a DL based algorithm PAF for the in-car scenario is presented. This work was first presented in [82].

Chapter 7 shows the development of a sub-toolchain for algorithmic qualitative and quantitative evaluation. The algorithm training and evaluation through the datasets generated with the developed toolchains is presented and discussed, as well as the main conclusions for it.

Chapter 8 presentes conclusions and outlining new directions for future work in the field.

Chapter 2

Tools Overview

This chapter discusses the hardware and software technologies used transversally throughout the sub-tasks. In-depth description is shown, as well as a requirement overview for each part.

Contents

2.1 Hardware Overview

In this section, the hardware systems used across the development pipeline are shown. The context of requirement for each chapter is described, as well an as in-depth description of each system.

2.1.1 Inertial System

Section 1.5.1 refers to the implementation and evaluation of the selected ground-truth system, a main requirement for the real dataset generation toolchain in Section 1.5.2. As it is shown in Chapter 4, one of the main parts of the ground-truth system is the inertial suit MVN Awinda [19]. The selection of the inertial-based ground-truth system was done concurrently with Bosch, considering the state-of-the-art overview presented in Chapter 1 and the technological limitations when applied in the in-car scenario. Robustness to occlusion, magnetic distortion and drift converged the decision to the MVN Awinda system.

The MVN Awinda is a suit comprised of 17 wireless inertial sensors capable of capturing each body segment orientation [9]. Through proprietary kinematic forwarding algorithms, the system is able to estimate specific skeletal joint 6 Degrees-of-Freedom (DoF) data wrt. its world axes. The entire inertial based motion capture system is processed in the MVN Studio software, where it is possible to calibrate the subject, monitor and record its body pose [19]. Sensor data is accessible through the MT Software Development Kit (SDK) [19] and body pose data is accessible through network protocol decoding [83], allowing for software synchronization. In regard to hardware synchronization, file format exporting is also available. Different solutions mean different types of data.

2.1.1.1 Motion Capture

In order to do human motion capture, MVN Awinda provides a modified version of the biomechanical model proposed by the Internal Society of Biomechanics (ISB) [53], with 23 segments and 22 joints [54]. Through the placement of 17 sensors in each segment of the human body following the MVN Awinda procedure [9] (Figure 2.1) the system estimates and captures the kinematic model of the human subject.

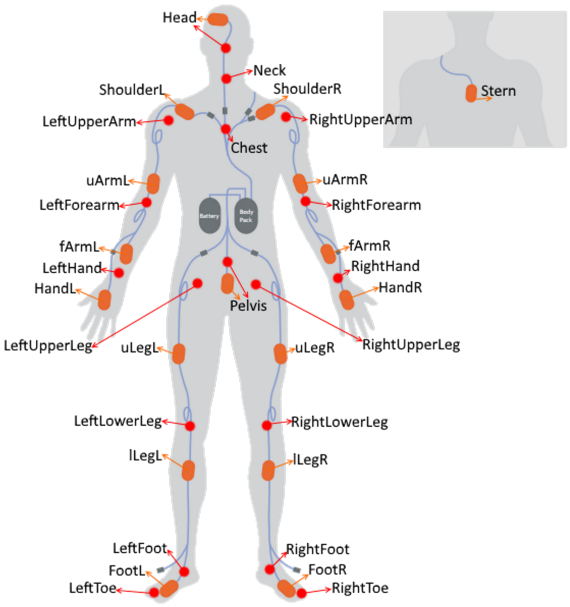


Figure 2.1: Body sensors' placement: orange represent the segment labels and red the joint labels. Adapted from MVN quick setup sheet [9].

The estimation is done through the MVN Studio software, using proprietary algorithms for body pose [84] and absolute positioning through Kalman Filter (KF). Once calibrated with the subject's body anatomical dimensions and initial body pose (i.e anatomical frame), the kinematic model is post-processed and captured in real time (equation 2.1 and Figure 2.2).

$$
\begin{aligned}
q &= q_r + q_i\boldsymbol{i} + q_j\boldsymbol{j} + q_k\boldsymbol{k} \\
\therefore {}^{G}p_{landmark} &= {}^{G}\,\boldsymbol{p}_{origin} + {}^{GB}\,q \otimes {}^{B}\,\boldsymbol{x} \otimes {}^{GB}\,q^{*} \\
\therefore {}^{B_A B_B}\,q &= {}^{GB_A}q^{*} \otimes {}^{GB_B}\,q
\end{aligned}
\tag{2.1}
$$

Positions of anatomical landmarks with respect to the joint origin of the related segment $\boldsymbol{{}^{G}p_{landmark}}$ in the global frame $\boldsymbol{G}$ can be found by rotating the vector of the landmark $\boldsymbol{{}^{B}x}$ in the body frame $\boldsymbol{B}$ to the global frame and adding the global position of the origin. Joint rotation is defined as the orientation of a distal segment $\boldsymbol{{}^{GB_B}q}$ with respect to a proximal segment $\boldsymbol{{}^{GB_A}q}$. Where $\otimes$ denotes a quaternion multiplication, $*$ the complex conjugate of the quaternion, $\boldsymbol{{}^{GB}q_{seg}}$ represents the quaternion vector describing the orientation of the segment with respect to the global frame.

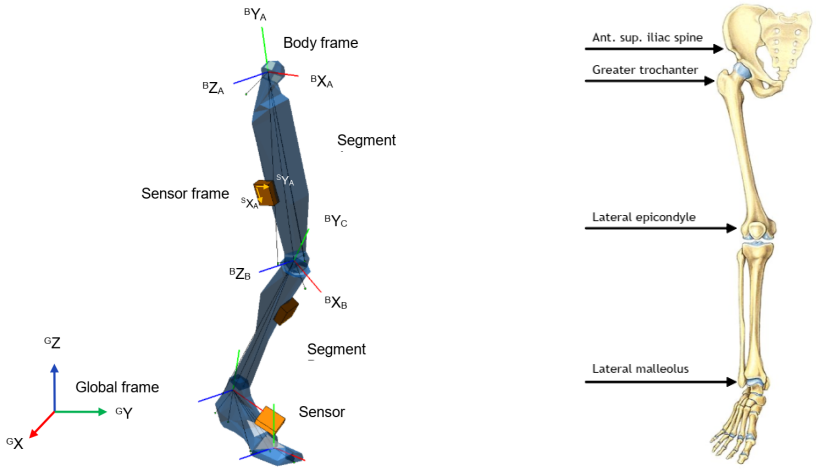


Figure 2.2: MVN Studio calculation of joint angles/positions: (A) global and local coordinate frames, and (B) body anatomical landmarks. Adapted from MVN quick setup sheet [9].

Available data is different depending on the final model and the type of access, as follows (Figure 2.3):

- MT SDK [19]: Sensor 3 DoF orientation data;
- Network protocol decoding [83]: Simplified ISB model (Figure 2.3B);
- Recorded file [19]: ISB model (Figure 2.3C).

Figure 2.3: MVN motion capture models: (A) live ISB model (joints and added markers/landmarks), (B) simplified ISB model (joints represented by rgb-axis) and (C) ISB model (joints and added markers/landmarks).

Considering the network access, the simplified ISB model is comprised of 20 skeleton joints' 6 DoF data (translation and rotation). In terms of recorded file access, the ISB model is comprised of the same 20 joints' 6 DoF data (translation and rotation) plus 64 body anatomical landmarks (e.g. pRightAuricularis). The ISB model relates itself with the human skeleton joints (Table 2.1).

Table 2.1: MVN models joint label correspondences. Virtual joints (i.e. the model does not perform direct joint estimation) are indicated in bold.

Human joints	**Simplified ISB (Index)**	**ISB**
Right Femur	$RightUpperLeg(15)$	$jRightHip$
Left Femur	$LeftUpperLeg(19)$	$jLeftHip$
Right Knee	$RightLowerLeg(16)$	$jRightKnee$
Left Knee	$LeftLowerLeg(20)$	$jLeftKnee$
Right Ankle	$RightFoot(17)$	$jRightAnkle$
Left Ankle	$LeftFoot(21)$	$jLeftAnkle$
Right Foot	$RightToe(18)$	$jRightToe$
Left Foot	$LeftToe(22)$	$jLeftToe$
Right Humerus	$RightShoulder(7)$	$jRightShoulder$
Left Humerus	$LeftShoulder(11)$	$jLeftShoulder$
Right Elbow	$RightUpperArm(8)$	$jRightElbow$
Left Elbow	$LeftUpperArm(12)$	$jLeftElbow$
Right Wrist	$RightForearm(13)$	$jRightWrist$
Left Wrist	$LeftForearm(9)$	$jLeftWrist$
Right Hand	$RightHand(10)$	$jRightTopOfHand$
Left Hand	$LeftHand(14)$	$jLeftTopOfHand$
Pelvis	$L5(0)$	$jL5S1$
T8 Vertebrae	$T8(4)$	$jT9T8$
C7 Vertebrae	$Neck(5)$	$jT1C7$
C1 Vertebrae	$Head(6)$	$jC1Head$

2.1.1.2 Software synchronization

Software synchronization can be used to capture synchronized information from several systems simultaneously. The MVN Awinda system allows this type of implementation through the MT SDK or network protocol decoding, using C++/.NET or even MATLAB.

Through the MT SDK, a 3^{rd} party application can connect directly to the MVN Awinda sensors and collect 3 DoF orientation data from each as illustrated in Algorithm 2.1.1. A first stage of **Connection** is required to enable data capture, followed by the **Data** capture itself with synchronization timestamps and data samples.

Algorithm 2.1.1 MT STREAM SDK

1: **Connection:**

2: $detectDevices.begin();$
3: $openPort();$
4: $enableRadio();$
5: **Data:**

6: $packet = mtwCallbacks[i] -> getOldestPacket();$
Timestamp
7: $packet -> timeOfArrival();$
Quaternion sample
8: $packet -> orientationQuaternion();$

On the other hand, network protocol decoding requires the selection of an User Datagram Protocol (UDP) Application Programming Interface (API) in order to comunicate with MVN Studio and decode its data streaming. After this, one can capture the simplified ISB model with per joint 6 DoF data (Algorithm 2.1.2). UDP method is not so straightforward, due to the fact that there are no SDK functions that give direct access to data samples. Instead, data is transmited in a single ***string*** communication packet [83] with the entire full body model information indexed, ***index***, sequentially, and with fixed size of bytes for each data type **segment id**$\rightarrow 4bytes$, **segment translation**$\rightarrow 4*3bytes$, **segment quaternion**$\rightarrow 4*4bytes$, reserving a total of 32 bytes for each segment.

Algorithm 2.1.2 MVN UDP SDK

1: **Connection:**
Set communication settings
2: $com.ReceiverAddr.sinport = htons(UDPport);$
3: $com.ReceiverAddr.sinport.s_addr = inet\_addr(IPport);$
4: $bind(com);$
5: **Data:**
Timestamp
6: $DecodeDataToTimestamp(com);$
Full Body Model sample
7: $DecodeDataToSegmentTranslation(com(index));$
8: $DecodeDataToSegmentOrientation(com(index));$

Data is not recorded by the MVN system, instead it is accessed by a 3^{rd} party application for recording purposes.

2.1.1.3 Hardware synchronization

MVN allows for hardware synchronization through the Xsens Awinda Station [85], where several systems can be electrically connected with a master-slave configuration. MVN Studio uses the hardware trigger to record the suits' motion capture data synchronously with other systems' software. Data is exported through standard formats, one of them being *.c3d files [86] with 64 anatomical landmarks. In this case, the ISB full body model is exported.

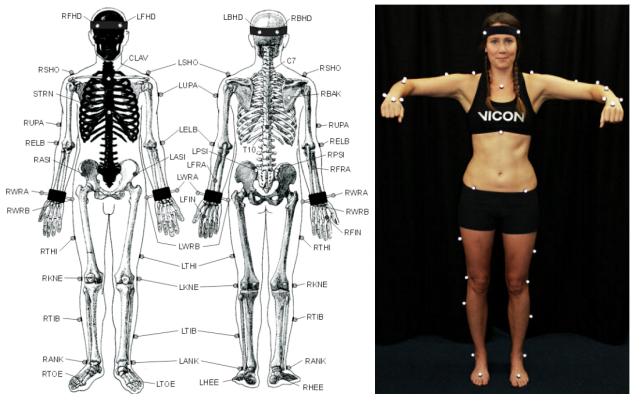
2.1.2 Vicon System

Like any other metrological evaluation, the assessment of the human motion capture system (Section 1.5.1) requires a better motion capture system to work as ground-truth. On the other hand, the ground-truth system requires an in-lab absolute positioning system to reference the ground-truth pose with image sensor devices. To satisfy both section requirements, the Vicon system is used. In this sense, the Vicon system is used to evaluate the MVN Awinda (Chapter 3), while also being used to obtain an absolute positioning for the in-lab scenario (Chapter 4).

The Vicon system is first and foremost a marker (IR passive/active) 3D tracking system. Through several marker placement configurations and specific software packages, the system is capable of capturing the human body pose as well as other custom objects. This is possible through the use of IR sensitive cameras that identify and triangulate the position of each visible marker. Through the Vicon Nexus software package, the system is able to do human motion capture of several human subjects at the same time, as well as custom made objects (custom configuration of markers). Data is accessible through DataStream SDK [87] for software synchronization solutions, or file format exporting for hardware synchronization solutions. Once again, each solution has access to different types of data.

2.1.2.1 Motion Capture

To perform human motion capture, Vicon provides the Plug-in Gait model [11]. Through the placement of 49 markers in the human body following Vicon's procedure [10] (Figure 2.4), it is possible for the system to estimate and capture the kinematic model of the human subject.



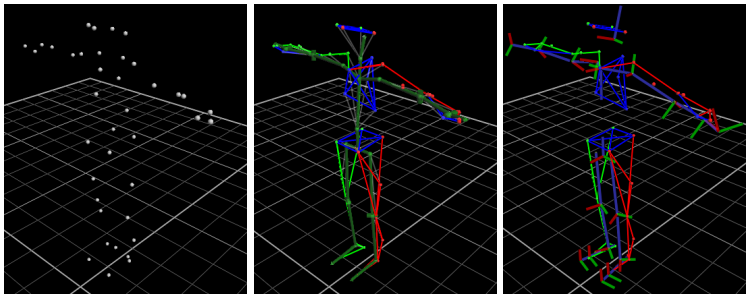
A B

Figure 2.4: Plug-in Gait marker placement: (A) guide, (B) human. Illustration used from Vicon Plug-in Gait marker placement manual [10].

The estimation is done through the Vicon Nexus software, using extensively the chord function [11] with

pre-inserted subject dimensions that were measured. Once calibrated, the kinematic model is captured in real time. As mentioned above, the output data depends on the final model and the type of access (Figure 2.5) as follows:

- DataStream SDK [87]: Kinematic Fit model (Figure 2.5B);
- Recorded file [88]: Plug-in Gait model (Figure 2.5C).



A B C

Figure 2.5: Vicon motion capture models: (A) raw marker data, (B) kinematic fit model and (C) plug-in gait model.

Considering the SDK access, the kinematic fit model is comprised of 20 joint 6 DoF data (translation and rotation). In terms of recorded file access, the plug-in gait model is comprised of 19 joint 6 DoF data (translation and rotation). Although both models are similar wrt. human body limbs, they differ from the torso to the head. These differences come from the fact that the models do not completely follow the skeletal standard, giving the best estimation to the limb joints (Figure 2.6) and creating virtual joints for the rest of the body.

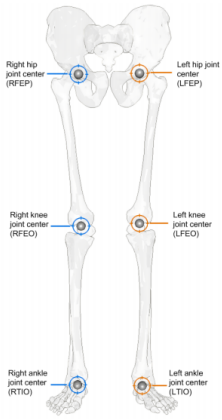


Figure 2.6: Plug-in gait model limb joints. Illustration used from Vicon Plug-in Gait manual [11].

Human body joints association can be seen in Table 2.2. Vicon's kinematic fit or plug-in gait models are not as realistic as the ISB model from MVN Awinda. Note also that, although the kinematic fit model

is comprised of segments and joints in the Vicon Nexus software, the DataStream SDK only gives access to segment data. This consideration does not constrain the evaluations as segment data is related with the segment's orientation 3 DoF and segment's origin position (joint) 3 DoF, therefore only requiring to relate the segment with the joint itself when joint data is needed.

Table 2.2: Vicon models joint label correspondence. Virtual joints (i.e. the model does not perform direct joint estimation) are indicated in bold.

Human joints	**Kinematic Fit**	**Plug-in Gait**	**Human joints**	**Kinematic Fit**	**Plug-in Gait**
Right Femur	R_Femur	$RFEP$	Right Elbow	R_Elbow	$RHIJ$
Left Femur	L_Femur	$LFEP$	Left Elbow	L_Elbow	$LHIJ$
Right Knee	R_Tibia	$RFEO$	Right Wrist	R_Wrist	$RRAO$
Left Knee	L_Tiibia	$LFEO$	Left Wrist	L_Wirst	$LRAO$
Right Ankle	R_Foot	$RTIO$	Right Hand	R_Wrist_End	$RHAO$
Left Ankle	L_Foot	$LTIO$	Left Hand	L_Wrist_End	$LHAO$
Right Foot	R_Toe	$RFOO$	Pelvis	***Root***	***PELO***
Left Foot	L_Toe	$LFOO$	T8 Vertebrae	***LowerBack_L_Collar***	***TRXO***
Right Humerus	$R\_Humerus$	RCJ	C7 Vertebrae	***LowerBack_Head***	
Left Humerus	$L\_Humerus$	LCJ	C1 Vertebrae	***Head***	***HEDO***

2.1.2.2 Software synchronization

The Vicon system allows for software synchronization through the DataStream SDK, using C++/.NET or even MATLAB. Through the DataStream SDK, a 3^{rd} party application can connect to the Vicon system (locally or through an IP address) and collect 6 DoF data from a custom object to a full body model as shown in Algorithm 2.1.3.

Algorithm 2.1.3 DATASTREAM SDK

Connection:

$MyClient.Connect(IP)$;
$MyClient.EnableSegmentData()$;
$MyClient.SetStreamMode(ServerPush)$;

Data:

for each s in $Segments$ **do**
# Get Timestamp
$MyClient.GetTimecode()$;
Get sample
$MyClient.GetSegmentGlobalTranslation(SubjectName, SegmentLabel)$;
$MyClient.GetSegmentGlobalRotationQuaternion(SubjectName, SegmentLabel)$;
end for

Data is not recorded by the Vicon system, instead it is monitored in real time by it, and accessed by a

3^{rd} party application for recording purposes. As it was mentioned before, for the purpose of human body motion capture, the kinematic fit is the only model accessible.

2.1.2.3 Hardware synchronization

Similarly to the MVN Awinda system, Vicon also allows hardware synchronization through the Lock+ system [89], where several systems can be electrically connected with a master-slave configuration. This configuration allows for a hardware trigger, where the system records its data in the proprietary software (Vicon Nexus) synchronously with the other systems software. Data is then exported through standard formats such as *.c3d files [86], where all motion data related to the subject for each timestamp is stored. In regard to the Vicon motion data, it represents the plug-in gait model.

2.1.3 ToF Camera

Section 1.5.2 refers to the real dataset generation using ground-truth data as well as Time-of-Flight (ToF) images, and Section 1.5.5 refers to a ToF image sensor for the online qualitative evaluation. As it is shown in chapters 4 and 7, image data is captured by a ToF sensor. After a state-of-the-art sensor evaluation and selection and considering Bosch roadmap, several iterations of ToF sensors were used in Chapter 4. The latest ToF sensor iteration was also implemented into the algorithmic evaluation toolchain (Chapter 7) for qualitative evaluations.

Image sensor requires in-depth understanding (i.e. metrological characteristics, image features, and automotive standards) as it is an important factor for the real dataset generation. It is necessary to quantify the measurement needs of the system being evaluated, it being the human body pose detection algorithmic system. This system resorts to image processing techniques and technology, so in this case we have several types of technology at our disposal to make image capture and processing. Because we are talking about image processing to obtain the position of each human body joint, it is necessary to capture the 3D space, including the subject. So, it is possible to use stereoscopic image processing or ToF (Figure 2.7).

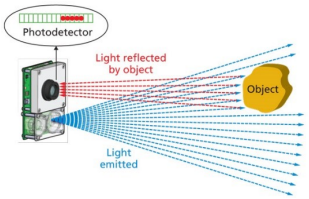


Figure 2.7: ToF principle. Illustration adapted from [12].

The sensor's resolution is one metrological characteristic inherent to both technologies that affects the

ground-truth system accuracy. Stereoscopic systems use standard RGB cameras that at present capture images at 1920x1080 pixels without adding great cost, making it possible to obtain spatial resolutions in the order of the millimeter, contrarily to the ToF system that has much inferior pixel resolutions. Despite this limitation, the ToF system still shows several advantages compared to the stereoscopic one, namely a bigger ambient light immunity, and the ability to capture the raw 3D space without the need for image processing (except some types of filters to smooth the 3D point cloud). With the Light-Emitting Diode (LED) evolution, it was possible to implement them in ToF systems [12], managing a substantial price reduction for this technology and at the same time making it more accessible for investigation projects. Nontheless, it still lacks the maturing seen in RGB cameras' pixel resolution.

ToF cameras estimate the 3D space through one of the following main modulation methods:

- Continuous Wave (CW): Measures the phase shift, $\boldsymbol{\phi}$, between the emitted, $\boldsymbol{s(t)}$, and received, $\boldsymbol{r(t)}$, signal (Figure 2.8A);
- Pulse Modulation (PM): Duration of the emitted pulse is set to the round trip time from light source to the further object (Figure 2.8B and equation 2.4).

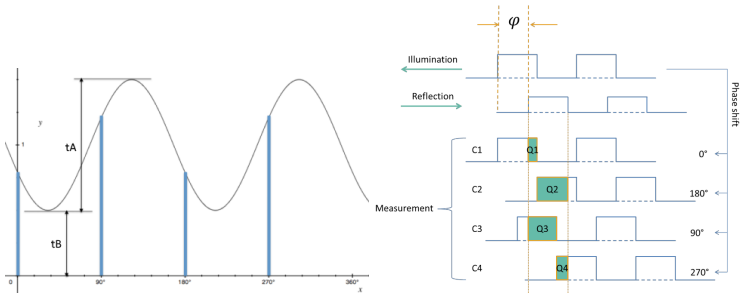


Figure 2.8: ToF principle: (A) CW, where blue lines represent each $\boldsymbol{C(x)}$ sample; (B) PM, where blue areas represent each $\boldsymbol{C(x)}$ sample.

Looking in-depth for the CW modulation [90] (equation 2.2), the ToF system is able to estimate the pixel-object distance, $\boldsymbol{d}$, through the cross-correlation, $\boldsymbol{C(x)}$, between emitted, $\boldsymbol{s(t)}$, and received, $\boldsymbol{r(t)}$, signals. Where $\boldsymbol{tA}$ represents the amplitude of the received signal and it depends on the object's reflectivity and sensor's sensitivity, $\boldsymbol{tB}$ represents the offset coefficient due to the ambient illumination, $\boldsymbol{f_m}$ represents the modulation frequency, and $\boldsymbol{\tau}$ represents the time of flight for light sent from the camera and received by it after being reflected by the object. $\boldsymbol{tA}$ and $\boldsymbol{tB}$ are unknown and are estimated at the camera.

$$
\begin{aligned}
s(t) &= a\cos(2\pi f_m t) \\
r(t) &= tA\cos(2\pi f_m(t-\tau)) + tB \\
\therefore C(\varkappa) &= \frac{a.tA}{2}\cos(2\pi f_m \varkappa + 2\pi f_m \tau) + tB
\end{aligned}
\tag{2.2}
$$

Estimation is possible through the **four bucket method** (Figure 2.8A and equation 2.3). Four equidistant samples of the received signal, $\boldsymbol{r(t)}$, within one cycle are captured, where the cross-correlations with the sent signal, $\boldsymbol{s(t)}$, are $\boldsymbol{c(\varkappa_0), c(\varkappa_{90}), c(\varkappa_{180}), c(\varkappa_{270})}$. Where $\boldsymbol{lc}$ is the light speed constant. Also it is important to observe that $\boldsymbol{\phi}$ is a phase wrapping because it ranges from $\boldsymbol{[0, 2\pi]}$, limiting the maximum distance, $\boldsymbol{d_{max}}$, with it.

$$
\begin{aligned}
tA &= \frac{\sqrt{[c(\varkappa_{270}) - c(\varkappa_{90})]^2 + [c(\varkappa_0) - c(\varkappa_{180})]^2}}{2} \\
tB &= \frac{c(\varkappa_0) + c(\varkappa_{90}) + c(\varkappa_{180}) + c(\varkappa_{270})}{4} \\
\tau &= \arctan\left[\frac{c(\varkappa_{270}) - c(\varkappa_{90})}{c(\varkappa_0) - c(\varkappa_{180})}\right] \\
\phi &= 2\pi f_m \tau = 2\pi f_m \frac{2d}{lc} \\
\therefore d &= \frac{\tau lc}{2}
\end{aligned}
\tag{2.3}
$$

Looking in-depth for the PM [17], light modulation is different but the principal is the same (equation 2.4).

$$
\begin{aligned}
Q1 = c(\varkappa_0), Q2 = c(\varkappa_{90}), Q3 = c(\varkappa_{180}), Q4 = c(\varkappa_{270}) \\
\phi = tan^{-1}\left(\frac{Q4 - Q3}{Q1 - Q2}\right) \\
\therefore d = \frac{c}{2f_m}\frac{\phi}{2\pi}
\end{aligned}
\tag{2.4}
$$

Breuer et al. made a depth sensor comparison and accuracy analysis [91], focusing more on the Kinect V2 accuracy and precision in standard scenarios. The sensor showed significant distance dependent error, requiring further calibration. The warm up experiment suggested a temperature dependent bias that needs to be corrected. Integration over multiple raw frames offered a measuring range of up to 18 meters. Analysis of pixel noise showed a strong relation to non-correlated scene illumination. The high resolution RGB camera provides a second image for triangulation, offering an extended range and higher accuracy in short range measuring.

Affordable systems [13] (Figure 2.9A) show higher pixel resolution of 512x424 pixels and a range of 50 centimeters to 4,5 meters. This type of technology has had a great reception in the field of investigation,

mostly for 3D motion capture. Wei et al. [92] developed a method of capture and reconstruction of the human body movement in 3D space, using the depth information provided by Kinect, concluding that the system is viable for real time virtual models control with great accuracy, using a real person's movement. Unfortunately, this technology is not normalized to be used inside car interiors, mostly because of its laser emissions, where Kinect only complies with International Electrotechnical Commission (IEC) 60825-1:2007 (Safety of laser products – Part 1: Equipment classification and requirements) for a Class 1 laser product, not focusing on several other automotive standards.

TI (Texas Instruments Inc., Texas, United States)) provides an evaluation kit for the OPT8241 ToF sensor (Figure 2.9B), with a resolution of 320x240 pixels, $[12; 60]$ frames per second (fps), and an operating range up to $4\ m$ (dependent on the modulation frequency). On the software side, there is the Voxel-SDK for 3rd party development, and the Voxel Viewer (built on top of Voxel SDK) to configure/calibrate/monitor the kit. The evaluation kit supports several output formats: Phase ,$\boldsymbol{C(\varkappa)}$, amplitude, $\boldsymbol{tA{\equiv}amp_{x,y}}$, depth, $\boldsymbol{d{\equiv}depth_{x,y}}$, and point cloud, $\boldsymbol{pc_f}$. Embedded temporal and spatial filtering is also possible. Unfortunately, this evaluation kit is not certified for automotive products, where the only standard provided by the Infrared (IR) emissions is Class 1 Laser Product European Standard (EN)/IEC 60825-1 2007&2014 (similar to Kinect v2).

SICK (Sick AG, Waldkirch, Germany) is a company known for its industrial sensors, and recently they developed a ToF sensor (Figure 2.9C) for industrial applications, more precisely to aid on Artificial Intelligence transporters motion. The principle is the same of other ToF sensors, but with less resolution (176x144 pixels) at 30 fps, with a Gigabit Ethernet interface and an API for easy development in Java, Python, C++, C#, etc. It also allows to filter, reduce and manipulate streamed data inside the device (embedded programming). Unfortunately, this is a certified industrial product that can be considered a closed box and is not prepared nor certified to be part of an automotive product. Still, its IR emissions are certified as EN 62471:2008 – "Risk Group 0".

A B C

Figure 2.9: ToF image sensors: (A) representation of point cloud from Kinect V2 comercial product (Illustration adapted from [13]); (B) Texas Instruments OPT8241 (Illustration adapted from [14]); (C) SICK V3S110 (Illustration adapted from [15]).

Sensor selection was done bilaterally taking into consideration Bosch sensor roadmap, shifting the focus to ToF image sensors. Through the years of 2016 to 2018 three product iterations were made and used: (1) Melexis (Melexis, Ypres, Belgium) EVK75023, (2) Melexis EVK75123 and (3) Gene8 Pico Monstar 105. While preserving automotive standards and improved image data quality, there was always the need to achieve higher Field-of-View (FoV) due to the implicit cockpit visibility requirement as illustrated in Figure 2.10.

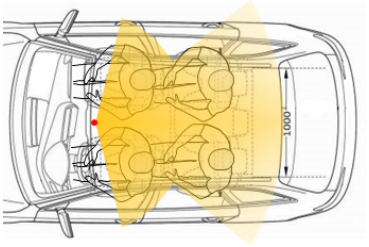


Figure 2.10: Bosch roadmap for ToF sensor placement and visibility requirements.

2.1.3.1 Melexis EVK75023

In the in-car scenario, it is necessary to use a ToF system that is normalized, as in the case of EVK75023 (Figure 2.11) from Melexis which complies with several automotive standards:

- International Organization for Standardization (ISO)/Technical Specifications 16949:2009 Quality management systems – Particular requirements for the application of ISO 9001:2008 for automotive production and relevant service part organizations;
- Automotive Electronics Council-Q100 Failure Mechanism Based Stress Test Qualification for Integrated Circuits;
- EN 62471:2008 Photobiological safety of lamps and lamp systems:
  - LED-Lens FL70 – "Risk Group 0 - Exempt Group" no warning labels needed;
  - LED-Lens FL66s – "Risk Group 1" warning labels needed;
  - LED-Lens FL63s – "Risk Group 1" warning labels needed;
  - LED-Lens FL90 – "Risk Group 1" warning labels needed.

In the case of IR light emissions, the "Risk Group 1" LED's need to be labeled to warn about the maximum eye exposure time for a distance of $20\ cm$ to it. Although Melexis provides several standards on its supply chain, it still represents a tier 2 supplier, where they can only guarantee the standard for the ToF sensor and LED IR emitters. In order for the final product to be fully compliant with the automotive standards, Bosch

has the responsibility to make the system automotive ready. This is true because Bosch is the tier 1 supplier and needs to qualify each individual component of the system.

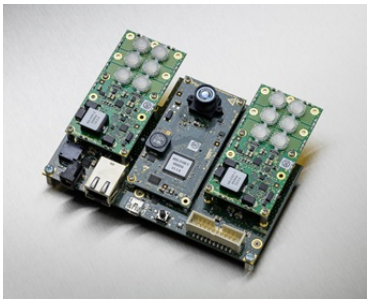
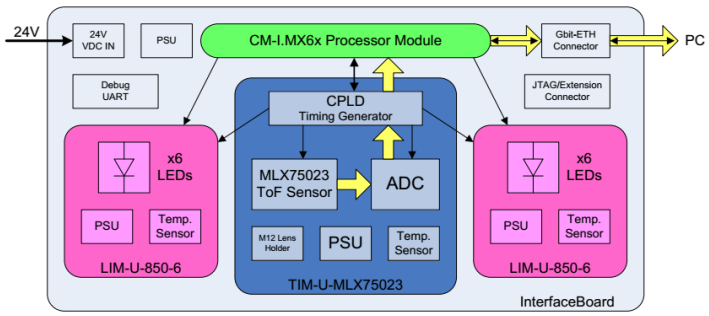


Figure 2.11: Melexis EVK75023 ToF sensor. Illustration adapted from [16].

The evaluation kit is comprised of two LIM-U850-6 (Illumination Unit) equipped with LED-Lens FL70, one TIM-U-MLX75023 (ToF Sensor Board), one CM-i.MX6x (Processor Module), and one interface board (Figure 2.12A). In regard to the ToF sensor, it only has a pixel resolution of 320x240 pixels and an application range of $3.7\ m$ to $1.6\ km$ (dependent of the modulation frequency in Figure 2.12B), which means that in a scenario of $2.5\ m$ of height (necessary for a full human body capture) it will give us a theoretical spatial resolution of approximately $10\ mm/pixel$. However, such resolution is not truly achievable because this type of technology only has a depth accuracy of centimeters. It is important to point out that the spatial resolution mentioned before is relative to the X and Y coordinates, because the Z axis (depth) requires further testing to be quantified.



Frequency	Exact frequency	Ambiguity Range
0,09 MHz	0,091463 MHz	1638873 mm[1)]
0,1 MHz	0,1 MHz	1498962 mm[1)]
0,11 MHz	0,11194 MHz	1339077 mm[1)]
1 MHz	1,2 MHz	124914 mm[1)]
5 MHz	5,2 MHz	28826 mm
7.5 MHz	7,7 MHz	19467 mm
10 MHz	10,2 MHz	14696 mm
15 MHz	15,2 MHz	9862 mm
20 MHz	20,2 MHz	7421 mm
25 MHz	25 MHz	5996 mm
30 MHz	30,2 MHz	4963 mm
35 MHz	35,169903 MHz	4262 mm
40 MHz	40,2 MHz	3729 mm

A B

Figure 2.12: EVK75023: (A) Block diagram, and (B) range vs modulation frequency. Illustrations adapted from [16].

The data and control interface is done via Gigabit-Ethernet, where the control interface is used to set and read configuration registers, allowing several types of configurations, such as:

- Calibrations: temperature compensation, distance offset related to modulation frequency, lens calibration, illumination compensation, modulation frequency, frame average, and integration time;

- Filters: median, bilateral, sliding average, wiggling compensation, and Fixed-Pattern-Phase Noise compensation.

The data interface provides a continuous stream of raw phase, amplitude, depth and point cloud data:

- Amplitude frame $\boldsymbol{tA \equiv amp_{x,y}}$: Reconstructed from the sum of raw phase data of all 4 phases $\mathbf{0, 90, 180, 270}$, and is associated with the material reflectivity to the IR light emitted.
- Depth frame $\boldsymbol{d \equiv depth_{x,y}}$: Is associated with the distance between the pixel and the object that is pointing to. The distance of each pixel is coded in milimeters;
- Point cloud matrix (XYZ voxel) $\boldsymbol{pc_f}$: Is associated with the cartesian position of each pixel.
- Balance: Is associated with the confidence of depth measurement of each pixel.

The evaluation kit comes with BltTofApi SDK that allows development in C, C++, C# and MATLAB, giving enough freedom for 3^{rd} party toolchain implementation. SDK allows for software synchronization and with a simple pipeline it is possible to connect and capture data from the sensor as illustrated in Algorithm 2.1.4. BltTofApi is available for x86/x64 and Advanced RISC Machine architectures, meaning that higher level applications can be deployed natively onto the ToF device itself with less effort. The developer has access to a dedicated partition for user applications, embedded on the ToF device Main Flash. This is possible through SSH connection, giving access to '/mnt/user' directory. The device can be programmed with the use of the Bluetechnix ToF API (BTA), although the API runs exclusively on the i.MX6 Quad processor (four Cortex-A9 cores at 1GHz). The Graphics Processing Unit (GPU) access is also possible through Open Computing Language. Lower level access to the Central Processing Unit (CPU) is also possible through NEON, a set of Single Instruction Multiple Data instructions.

Algorithm 2.1.4 BltTofApi SDK

```
Connection:
BTAinitConfig(&config);
BTAopen(&config);
Data:
BTAgetFrame(&frame);
# Timestamp
frame->timeStamp;
# Frame samples
# Amplitude
BTAgetAmplitudes(frame);
# Point cloud
BTAgetXYZcoordinates(frame);
# Depth
BTAgetDistances(frame)
# Balance
frame.channels.data
```

In terms of image quality control, the EVK75023 uses an IR exposure confidence level for each pixel. The exposure is not of the same quality for each pixel, and that leads to a noisy image. An extra frame is accessible, $\boldsymbol{confidence}$, giving the confidence level of each pixel with embedded filtering, with the depth information for pixels with low confidence not being set. As mentioned before, several filters are accessible. Temporal filtering is a median FIFO stack, $\boldsymbol{FIFO_{stacks}}$, with a maximum number of 15 frames, that allows to reduce the gaussian noise at the expense of reducing the frame rate (i.e. $\boldsymbol{f_{out} = \frac{f_{in}}{FIFO_{stacks}}}$) and added motion blur. Spatial filtering is also available through a fixed size averaging kernel $\boldsymbol{W \cdot H}$ that makes several frame passes. In Figure 2.13 it is possible to see temporal and spatial filtering applied to the evaluation kit's output data.

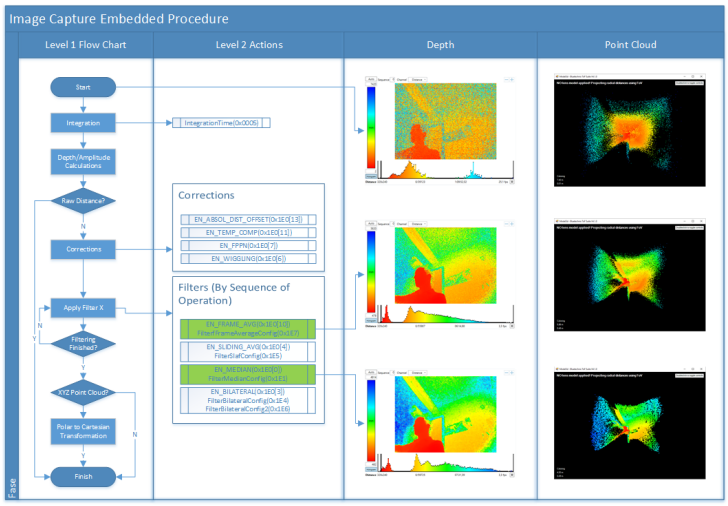


Figure 2.13: EVK75023 image filtering experimentation. Depth and point cloud are shown for each sequential filtering configurations: (top) no filtering; (middle) temporal filtering; (bottom) temporal and spatial filtering.

It is important to understand that post-processing filtering (temporal and spatial) does not increase the Signal-to-Noise Ratio (SNR). To achieve a better SNR, an increased integration time is needed, although this factor increases the system's temperature and eye safety concerns.

2.1.3.2 Melexis EVK75123

Later in 2017, the new Melexis EVK75123 was released (Figure 2.14), bringing the same automotive standards but with improved characteristics, such as:

- Smaller form factor 80x50 mm;
- Programmable illumination power with higher peak power;
- Illumination from LED to Vertical-Cavity Surface-Emitting Laser, allowing higher modulation frequencies and better SNR;

- FoV increased to 110, although reduced to 100 due to barrel distortion;
- Identical SDK interface.

Figure 2.14: Melexis EVK75123 ToF sensor. Illustration adapted from [17].

2.1.3.3 Gener8 Pico Monstar 105

In 2018, a new ToF iteration was possible, with the same automotive standards as the Melexis products, but with a more closed box concept (similar to the likes of Kinect V2). The Pico Monstar 105 comes with increased pixel resolution 352x287 and similar FoV of 100x85. Although lacking the lower level configurations as filtering, modulation frequency, integration time, calibration, etc, it gives access to specific use-cases with predefined embedded configurations (Table 2.3).

Table 2.3: Pico Monstar 105 use-cases.

Use Case	**Range(m)**	**FPS (Hz)**	**Integration Time (μs)**
1	$1.0 - 4.0$	5	2000
2	$1.0 - 4.0$	10	1000
3	$0.5 - 1.5$	15	700
4	$0.3 - 2.0$	25	450
5	$0.3 - 2.0$	35	600
6	$0.1 - 1.0$	45	500

SDK support comes in the form of libroyale [93], allowing 3^{rd} party implementation in C, C++ and MATLAB (Algorithm 2.1.5). Image data is also similar to the Melexis EVK, although changing balance frame for confidence frame, and also adding a noise frame. These last two frames are not required for the dataset generation, although they can be used for improved image data understanding.

Algorithm 2.1.5 Libroyale SDK

1: **Connection:**

2: $cameraDevice- > initialize();$
3: $cameraDevice- > setUseCase(\Gamma);$
4: $cameraDevice- > startCapture();$
5: **Data:**
Timestamp
6: $data- > timeStamp;$
Frame samples
Amlitude
7: $data- > points.grayValue;$
Point cloud
8: $data- > points.x;$
9: $data- > points.y;$
10: $data- > points.z;$
Confidence
11: $data- > points.depthConfidence;$
Noise
12: $data- > points.noise;$

2.1.4 Car TestBed

Sections 1.5.2 and 1.5.5 generate and evaluate data for the in-car scenario. This focus brings the need for a car testbed. As it is shown in chapters 4 and 7, a car testbed is prepared for the sole purpose of having a controlled in-lab environment for a functioning car interior. This allows for better real dataset generation as well as validation.

2.1.5 Computing Server

Sections 1.5.1 to 1.5.5 require computational capabilities for the different tasks at hand. Each chapter uses the same computing server, built to satisfy the requirements of the entire development pipeline:

- Chapter 3 - For the ground-truth evaluation several toolchains were developed, all needing real time recording from multiple systems. This requirement adds the need for high CPU core number, disk write/read speed, disk storage and Random-Access Memory (RAM) storage;
- Chapter 4 - Real dataset generation has the same requirements of the previous chapter;
- Chapter 5 - Synthetic dataset generation as the same requirements, but due to different needs. The developed toolchain does not use external systems and relyes extensively on the server resources for an high throughput of synthetic data generation;
- Chapter 6 - Algorithmic development does not rely on the CPU requirements like in other chapters. Machine Learning (ML) based methods training is supported by GPU arquitectures based on Compute

Unified Device Architecture cores, due to inherited instruction paralelism. New GPU server arquitectures bring improved resources compared with standard desktop, as well as new architectures for ML focused instructions. One of those improvements is the hardware level matrix operations per clock cycle (Tensor Cores [18]) as illustrated in Figure 2.15, giving up to 12x performance gains compared with standard single 16 bit floating point operations.

- Chapter 7 - Algorithmic evaluation needs dataset storage and algorithmic infering, to which adds the requirements of all previous chapters.

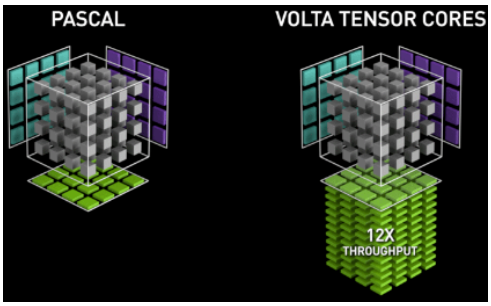


Figure 2.15: Nvidia tensor core performance illustration. Image addapted from [18].

Considering the development requirements and project funds, a custom Fujitsu server (Figure 2.16) was used with the following caracteristics: Xeon Gold 6140 18 cores 2.3GHz; 128GB DDR4 2.6GHz RAM; Nvidia V100 16GB High Bandwith Memory 2, Peripheral Component Interconnection Express; 480GB Solid-State Drive (SSD) SATA6; and 2TB 10Krpm Hard Disk Drive.

Figure 2.16: Fujitsu RX2560 M4 server for computing requirements throughout the development pipeline.

2.2 Software Overview

In this section, the software systems used across the development pipeline are shown. The context of requirement for each chapter is described, as well as an in-depth description.

2.2.1 ADTF

Chapters 3 and 4 have several toolchains that require real time recording from multiple systems, which can be performed through software synchronization. Automated Data and Time-Triggered Framework (ADTF) [94] is specifically focused for real time software synchronized data aquisition pipelines.

ADTF is a framework for synchronized data capture, process, monitor and actuation between different systems. Each specific higher level function (i.e. system data capture) is implemented in a custom made C++ based function-block, named Filter. Each filter follows an implementation protocol (Figure 2.17A shows the main implementation protocol, with a sub-protocol for the $\boldsymbol{\_Filter :: Init()}$ and $\boldsymbol{\_Filter :: Shutdown()}$ functions being shown in Figure 2.17B), in order to be used by ADTF, making it possible for each filter to declare IOs $\boldsymbol{Init(StageFirst)}$, receive user data $\boldsymbol{\Gamma}$ (Figure 2.17C) from ADTF $\boldsymbol{Init(StageNormal)}$, boot and start its task $\boldsymbol{\_Filter :: Start()}$.

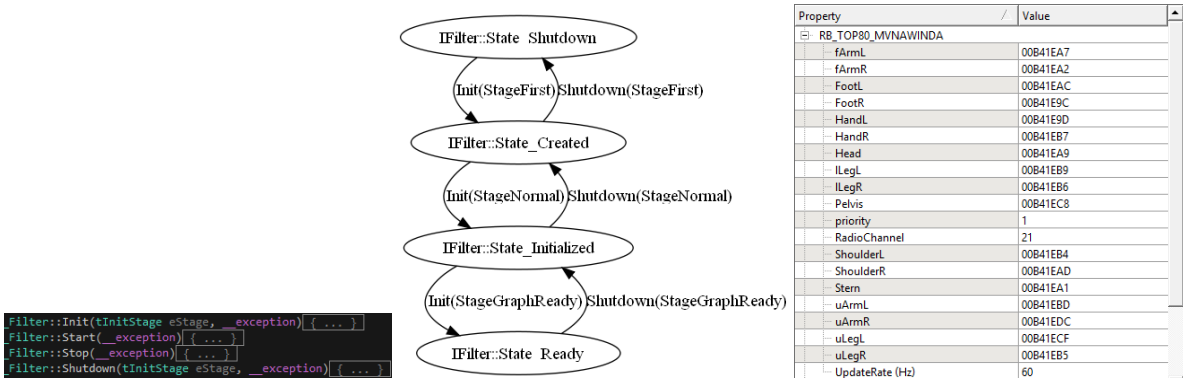


Figure 2.17: ADTF implementation: (A) protocol, (B) sub-protocol, and (C) user input $\boldsymbol{\Gamma}$ for a specific filter.

When a filter is fully implemented it can be seen as a black-box with inputs and outputs of different data types (e.g boolean, int, float, struct), where they represent the data captured/sent by/of hardware systems (e.g. images, positions, orientations, etc.). Upon developing all filters, an implementation with higher level of abstraction can be used in ADTF, where all filters are used in a pipeline to achieve the purpose of the tool itself (Figure 2.18).

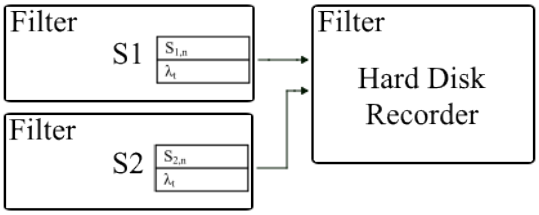


Figure 2.18: ADTF pipeline graph (e.g. two filters for two hardware systems [S_1 and S_2] sending data to a hard disk recording filter).

Once ADTF starts the entire pipeline (Figure 2.18), each filter will follow its internal booting sequence (Figure 2.17B) and interact with each other. Here is where ADTF excels, by giving synchronization microsecond timestamps $\lambda_t, t = 0, ..., T \; \mathbb{N} \cap [0, \infty]$ (T represents the number of recorded samples from all systems), for each data sample $S_{s,n}, n = 0, ..., N$ (N represents the number of samples for the filter) of each filter, s, as can be seen in Figure 2.19. These timestamps allow for robust data interpolation (e.g. inside a filter or outside ADTF), generating new data samples with identical timestamps for all filters.

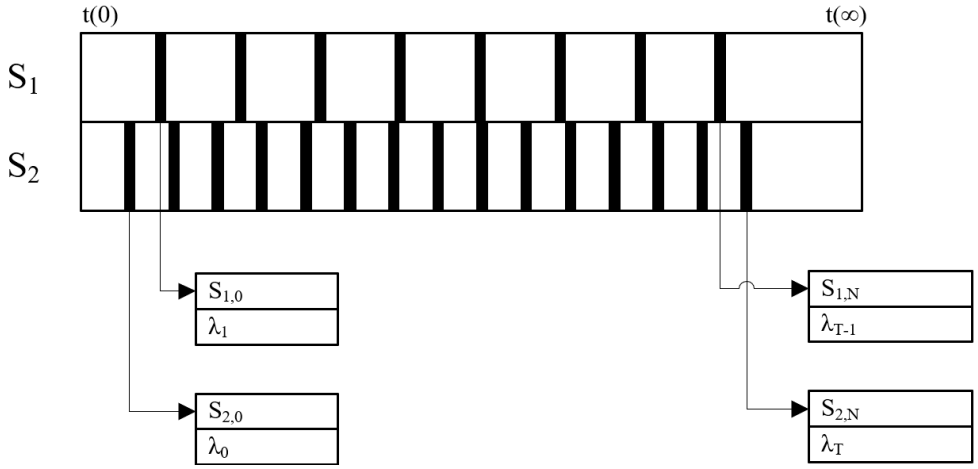


Figure 2.19: ADTF data recording synchronization graph. Example shows two filters generatinng samples $S_{s,n \rightarrow N}$ with different sampling frequencies, with associated global timestamps $\lambda_{t \rightarrow T}$

2.2.2 MATLAB

Chapters 3, 4 and 7 have several toolchains that require data post-processing to solve three main needs: (1) ADTF data import and time interpolation, (2) data representation and (3) error quantification. MATLAB is the main tool for this task, allowing for fast software prototyping.

MATLAB is a fast prototyping software development tool, giving the possibility to manipulate data in a fast way when processing speed is not the focus. In order to post-process ADTF recorded data, MATLAB needs to import it. Metadata files (*.description), that describe the output data from each recorded filter, are used in conjunction with each recorded file, to inform MATLAB about the data structure of each system (Figure 2.20).

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<adtf:ddl xmlns:adtf="adtf">
 <header>
  <language_version>3.00</language_version>
  <author>João Borges</author>
  <date_creation>01.02.2017</date_creation>
  <date_change>01.02.2017</date_change>
  <description>ADTF generated</description>
 </header>
 <datatypes>
  <datatype description="predefined ADTF tBool datatype" max="tTrue" min="tFalse" name="tBool" size="8" />
  <datatype description="predefined ADTF tChar datatype" max="127" min="-128" name="tChar" size="8" />
  <datatype description="predefined ADTF tFloat32 datatype" max="3.402823e+38" min="-3.402823e+38" name="tFloat32" size="32" />
  <datatype description="predefined ADTF tFloat64 datatype" max="1.797693e+308" min="-1.797693e+308" name="tFloat64" size="64" />
  <datatype description="predefined ADTF tInt16 datatype" max="32767" min="-32768" name="tInt16" size="16" />
  <datatype description="predefined ADTF tInt32 datatype" max="2147483647" min="-2147483648" name="tInt32" size="32" />
  <datatype description="predefined ADTF tInt64 datatype" max="9223372036854775807" min="-9223372036854775808" name="tInt64" size="64" />
  <datatype description="predefined ADTF tInt8 datatype" max="127" min="-128" name="tInt8" size="8" />
  <datatype description="predefined ADTF tUInt16 datatype" max="65535" min="0" name="tUInt16" size="16" />
  <datatype description="predefined ADTF tUInt32 datatype" max="4294967295" min="0" name="tUInt32" size="32" />
  <datatype description="predefined ADTF tUInt64 datatype" max="18446744073709551615" min="0" name="tUInt64" size="64" />
  <datatype description="predefined ADTF tUInt8 datatype" max="255" min="0" name="tUInt8" size="8" />
 </datatypes>
 <structs>
  <struct alignment="1" name="MVNAwinda" version="1">
   <element alignment="1" arraysize="17" byteorder="LE" bytepos="0" name="mediatype" type="tMVNAwinda" />
  </struct>
  <struct alignment="1" name="tMVNAwinda" version="2">
   <element alignment="1" arraysize="10" byteorder="LE" bytepos="0" name="sensorID" type="tUInt8" />
   <element alignment="1" arraysize="15" byteorder="LE" bytepos="10" name="bodySegmentID" type="tUInt8" />
   <element alignment="1" arraysize="7" byteorder="LE" bytepos="25" name="dummy1" type="tUInt8" />
   <element alignment="1" arraysize="4" byteorder="LE" bytepos="32" name="oneData" type="tFloat64" />
  </struct>
  <struct alignment="1" name="ViconNexus" version="1">
   <element alignment="1" arraysize="17" byteorder="LE" bytepos="0" name="mediatype" type="tViconNexus" />
  </struct>
  <struct alignment="1" name="tViconNexus" version="2">
   <element alignment="1" arraysize="10" byteorder="LE" bytepos="0" name="segmentID" type="tUInt8" />
   <element alignment="1" arraysize="15" byteorder="LE" bytepos="10" name="bodySegmentID" type="tUInt8" />
   <element alignment="1" arraysize="7" byteorder="LE" bytepos="25" name="dummy1" type="tUInt8" />
   <element alignment="1" arraysize="4" byteorder="LE" bytepos="32" name="oneData" type="tFloat64" />
  </struct>
 </structs>
 <streams>
  <stream description="streamid_1" name="MVNAwinda" type="MVNAwinda" />
  <stream description="streamid_2" name="ViconNexus" type="ViconNexus" />
 </streams>
</adtf:ddl>
```

Figure 2.20: ADTF example metadata file. $\boldsymbol{streamid\_1 = S_1}$ and $\boldsymbol{streamid\_2 = S_2}$.

In the illustration it is possible to see the memory padding of the entire data structure for each filter, where this information relates not only to the structure data types itself but also to the sequence of variable declaration. When a CPU arquitecture (x86, x64, etc) writes a datastructure into memory, it does so in a byte sized square shape to improve memory read/write performance. To fit a data structure into a square shaped memory layout, dummy bytes are added between variables when needed. This information is important to correctly read the recorded data from an ADTF recording. Using ADTF dat2mat proprietary script, it is possible to import the ADTF recorded data in MATLAB (Figure 2.21) making use of the metadata files. With a standard interface between ADTF and MATLAB, it is possible to manipulate data to satisfy the needs of the developed toolchains.

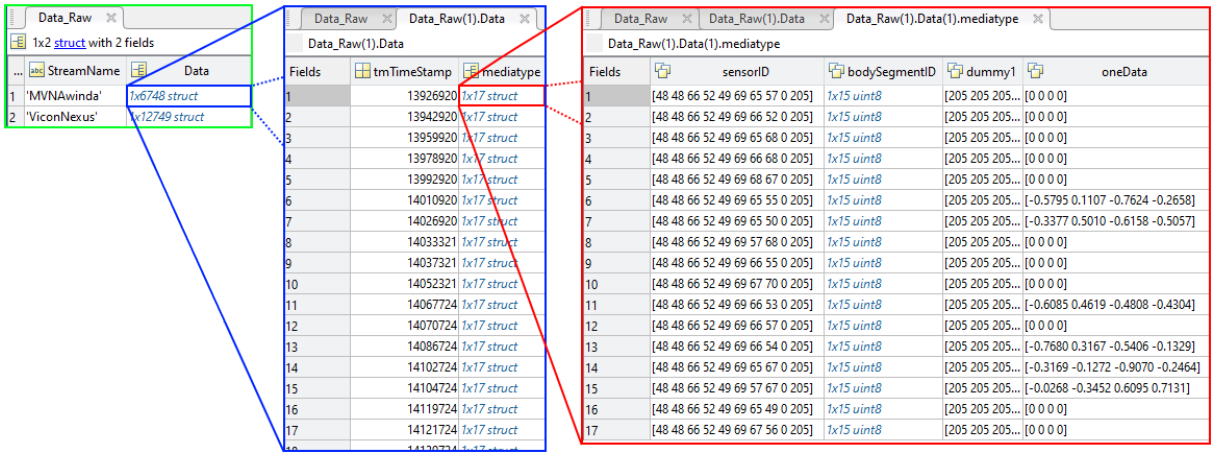


Figure 2.21: ADTF imported data in MATLAB: (green) all samples from $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$, (blue) all samples from $\boldsymbol{S_1}$ ($\boldsymbol{tmTimeStamp = \lambda_t}$ and $\boldsymbol{mediatype = S_{1,n}}$) and (red) all data from a single $\boldsymbol{S_{1,n}}$ sample.

2.2.3 Blender

Section 1.5.3 refers to the synthetic dataset generation. Chapter 5 is entirely focused on one single toolchain capable of simulating image sensors in a 3D scenario comprised by cars and humans, with human motion. To reduce the development time and complexity, the video game industry analogy can be applied. Instead of developing an entire game engine (in-car 3D scenario), a market solution is used as a base for development. Blender [95] is a great choice, allowing for all the main features that are requirements of the toolchain itself.

2.2.4 Python

As it was mentioned, Section 1.5.3 refers to the synthetic dataset generation, and Section 1.5.4 refers to the algorithm development. Although chapter 5 uses a base 3D engine (Blender), this engine does not deploy all the solutions for the synthetic toolchain. There is the need to build a custom toolchain that allows the user to customize image sensors, cars, human models, human motion and dataset. Blender is Python based, allowing for customization to the engine through Python custom scripts, making Python the obvious choice for toolchain development alongside Blender. Chapter 6 is less limited to development software choices, requiring different software tools depending on the available algorithms' implementations, as well as data preparation. Python is again an obvious choice, being used across most ML/Deep Learning (DL) frameworks, although being also a great tool for data preparation.

2.2.5 C++/C#

C++ is used not as a choice but as an implicit requirement from previous choices. ADTF is based in C++ software development (chapters 3 and 4), and, as it is shown in Chapter 6, the final algorithmic implementation is based in a C++ framework. In Chapter 5, a choice was made to use C# as the main programming language for Graphical User Interface (GUI) construction (to serve as a way to perform customizations within each toolchain) for toolchain customization.

Chapter 3

Ground-truth implementation and evaluation for the in-car scenario

This chapter considers the creation of a ground-truth system capable of capturing human body pose inside of a vehicle, in order to train and evaluate the algorithm in its development process. To satisfy the needs of the real dataset toolchain in Chapter 4, as well as the algorithm evaluation in Chapter 7, a selected ground-truth system was implemented and evaluated through an extensive evaluation procedure, with specific toolchains developed for this purpose. Preliminary versions of the work presented in this chapter were presented in the paper **"A system for the generation of in-car human body pose datasets"**, currently in revision process.

In the first section of this chapter, the evaluation procedures as well as the toolchains developed are shown. Four evaluation procedures were defined in order to better understand the ground-truth system sensor and full body model errors, both from a static and dynamic prespective. For each evaluation procedure, a toolchain was developed. Each toolchain presented hardware and software requirements, which are now shown in-depth.

In the second and third sections of this chapter, the evaluations results, discussion and conclusions are shown. Each of the four evaluation procedures present results that support a better understanding on the ground-truth system limitations and benefits. Finally, possible future improvements are then discussed.

Contents

3.1 Evaluation toolchains

In this section, the evaluation procedures and corresponding toolchains are shown. To best understand how the entire inertial suit behaves in human motion capture, there is a need to evaluate it in four distinct ways:

- Static evaluation:
  - Sensor (SSE) - understanding the sensor level sensitivity to magnetic distortions in each rotation axis;
  - Full body model (SFE) - understanding the full body joint's sensitivity to drift and magnetic distortions;
- Dynamic evaluation:
  - Sensor (DSE) - understanding the sensor level sensitivity to specific human body segments' rotation;
  - Full body model (DFE) - understanding the full body joints' translation error.

3.1.1 Static Evaluation

3.1.1.1 Static sensor evaluation

To understand how the inertial sensor is prone to error in each rotation axis, a toolchain had to be developed in order to rotate the sensor in each axis and compare its rotation with the real one. The toolchain used a nanotec SMCI47-S-2 stepper motor drive [96], to rotate a stepper motor in closed-loop, where the inertial sensor was mechanically coupled (500mm shaft coupling). Through the Nanotec Software Development Kit (SDK) [97] it is possible to rotate the stepper motor, $\boldsymbol{degrees}$, while capturing the sensor Euler rotation through the MT SDK [19] as shown in Algorithm 3.1.1.

Algorithm 3.1.1 SSE

```
Connection:
    # Sensor
    Connection in Alg.2.1.1
    # Motor
    motor = ComMotorCommands()
Data:

for each i in Niterations do
    motor.StartTravelProfile(degrees)
    for each s in Nsamples do
        mtwData._orientation.x()
        mtwData._orientation.y()
        mtwData._orientation.z()
    end for
end for
```

Equation 3.1 estimates the error of each sensor axis of rotation, $\boldsymbol{\xi_t}$. Calculating the deviation from the actual sensor axis orientation, $\boldsymbol{s\Theta_t}$, (without the initial offset $\boldsymbol{s\Theta_0}$) against the motor actual rotation, $\boldsymbol{m\Theta_t}$, where $\boldsymbol{s\Theta}$ represents an individual axis of rotation (roll $\boldsymbol{x}$, pitch $\boldsymbol{y}$ or yaw $\boldsymbol{z}$) depending on which is parallel with the motor axis rotation. This error allows us to know both the $\boldsymbol{s\mu}$ and $\boldsymbol{s\sigma}$ of the error considering all the recorded samples, where $\boldsymbol{nI}$ represents the motor rotation iterations ($\boldsymbol{nI \equiv Niterations}$) and $\boldsymbol{nS}$ the sensor samples in each 360^o rotation ($\boldsymbol{nS \equiv Nsamples}$).

$$
\begin{aligned}
s\Theta_t &= s\Theta_t - s\Theta_0 \\
\xi_t &= |s\Theta_t - m\Theta_t| \\
\therefore s\mu &= \frac{\sum_{t=0}^{nI \times nS} \xi_t}{nI \times nS} \\
\therefore s\sigma &= \sqrt{\frac{\sum_{t=0}^{nI \times nS} (\xi_t - s\mu)^2}{(nI \times nS) - 1}}
\end{aligned}
\tag{3.1}
$$

3.1.1.2 Static full body evaluation

Static full body evaluation was focused on understanding how the Internal Society of Biomechanics (ISB) model behaves in a real world driving scenario. Considering the fact that no other system can evaluate the inertial system inside a vehicle, the evaluation focused on comparing the initial N-Pose model before entering the vehicle and after driving. To achieve this evaluation a standard use of the MVN Studio is sufficient to extract both models joint's cartesian position and rotation at a specific timestamp (before entering the vehicle and after leaving it) in Excel format.

With such data, it is possible to compute the Euclidian error, $\boldsymbol{\xi_j}$, of each joint for any final N-Pose ($\boldsymbol{s = 2}$)

against the initial one ($\boldsymbol{s = 1}$) (equation 3.2), where $\boldsymbol{Pxyz_j}$ represents the 3D cartesian position for each of the 20 joints, $\boldsymbol{nJ}$.

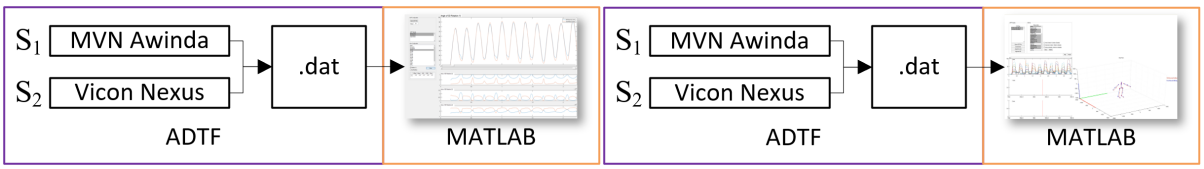
$$\begin{aligned} Pxyz_{s,j} &= \{x_{s,j}, y_{s,j}, z_{s,j}\} \\ \xi_j &= \sqrt{(x_{1,j} - x_{2,j})^2 + (y_{1,j} - y_{2,j})^2 + (z_{1,j} - z_{2,j})^2} \end{aligned} \tag{3.2}$$

Another important step to improve cartesian error estimation is the alignment of both N-pose frames in terms of rotation, $\boldsymbol{T_j^{pelvis}}$. Such alignment is required given that the subject does not stand in the N-Pose facing the same orientation in regard to the Earth magnetic north. To accomplish it, a root joint is selected (pelvis) and its transformation matrix, $\boldsymbol{T_{pelvis}}$, is used as a translation and rotation offset to the rest of the body joints (equation 3.3).

$$T_j^{pelvis} = (T_{pelvis})^{\intercal} \cdot T_j \tag{3.3}$$

3.1.2 Dynamic Evaluation

Each evaluation method (DSE and DFE) uses an independent toolchain due to its evaluation specificity. To perform a dynamic evaluation of the inertial suit, a more robust motion capture system is needed. Since this evaluation is focused in an in-lab environment, there is the possibility of using the golden-standard of human motion capture, the Vicon System. Also it is important to have a synchronized recording of both systems. Considering the fact that both allow software synchronization, the toolchains use software synchronization recording through Automated Data and Time-Triggered Framework (ADTF) [94]. Data post-processing and error quantification is done in a Graphical User Interface (GUI) implemention in MATLAB for each toolchain (Figure 3.1).



A B

Figure 3.1: Toolchain pipeline for dynamic evaluations: (A) DSE, and (B) DFE. MVN Awinda and Vicon Nexus Filters data (i.e. sensors and segments quaternions) are recorded synchronously in ADTF, and then post-processed in MATLAB for error quantification.

3.1.2.1 Dynamic sensor evaluation

To understand how the inertial sensor dynamic range behaves in each human body segment, each MVN Awinda sensor needs to be recorded synchronously with the Vicon kinematic-fit model and referenced to its corresponding segment, in order to compare both. Comparison is based on the raw quaternion data, $\boldsymbol{q_t}$, of each system (sensor for MVN Awinda and segment for Vicon kinematic-fit) sent from each filter. Each quaternion is then converted to axis-angle, $\boldsymbol{\theta_t}$, (equation 3.4), while removing the offset rotation of the first frame through the quaternion conjugate, $\boldsymbol{(q_0)^c}$, in order to have a relative rotation for each segment.

$$\begin{aligned} q_t &= q_{t,r} + q_{t,i}\boldsymbol{i} + q_{t,j}\boldsymbol{j} + q_{t,k}\boldsymbol{k} \\ qr_t &= q_t \cdot (q_0)^c \\ \theta_t &= 2 \times \arccos(qr_{t,r}) \end{aligned} \tag{3.4}$$

To achieve this, a toolchain was developed consisting of two parts: (1) synchronized recording through ADTF and (2) graphical quantitative evaluation through MATLAB.

Recording (ADTF)

First part requires the development of two distinct filters for data capture of MVN Awinda sensors, and Vicon kinematic fit model. Although distinct, there is a need to relate the filter output in order to later reference the sensor data with the segment data. For this, a standard output data structure is created, $\boldsymbol{S_{s,n}}$, where each sensor address or segment label is referenced (Table 3.1). Referencing is done as a filter initialization, although it is possible to customize the information in ADTF before boot, $\boldsymbol{\Gamma}$.

Table 3.1: Reference between sensor address and kinematic-fit segments.

$S_{s,n}$	Segment	Sensor	$S_{s,n}$	Segment	Sensor
Head	Head	00B41EA9	uLegL	L_Femur	00B41ECF
ShoulderL	L_Collar	00B41EB4	uLegR	R_Femur	00B41EB5
ShoulderR	R_Collar	00B41EAD	lLegL	L_Tibia	00B41EB9
uArmL	L_Humerus	00B41EBD	lLegR	R_Tibia	00B41EB6
uArmR	R_Humerus	00B41EDC	FootL	L_Foot	00B41EAC
fArmL	L_Elbow	00B41EA7	FootR	R_Foot	00B41E9C
fArmR	R_Elbow	00B41EA2	Stern	LowerBack	00B41EA1
HandL	L_Wrist	00B41E9D	Pelvis	Root	00B41EC8
HandR	R_Wrist	00B41EB7			

With this information, each filter is able to use the same output data structure $\boldsymbol{BodySensors}$ and $\boldsymbol{BodySegments}$ (Figure 3.2).

```
//structure that contains all the data related to each sensor
typedef struct tMotionSensor {
    tChar sensorID[10];         //address
    tChar bodySegmentID[15];    //body segment name
    tFloat64 oneData[4];        //quaternion sample (x,y,z,w)
}tMotionSensor;
//structure that will allow for the creation of a buffer of synced data to be sent to ADTF
typedef struct tMotionBodySensors {
    tMotionSensor BodySensors[sensor::count];
}tMotionBodySensors;
```

A

```
//structure that contains all the data related to each segment
typedef struct tMotionSegment {
    tChar segmentID[10];        //label
    tChar bodySegmentID[15];    //body segment name
    tFloat64 oneData[4];        //quaternion sample (x,y,z,w)
}tMotionSegment;
//structure that will allow for the creation of a buffer of synced data to be sent to ADTF
typedef struct tMotionBodySegments {
    tMotionSegment BodySegments[segments::count];
}tMotionBodySegments;
```

B

Figure 3.2: Filters output for DSE: (A) MVN Awinda, (B) Vicon Nexus.

Although the structure data type is different ($\boldsymbol{tMotionSensor \neq tMotionSegment}$), the relevant data format is similar $\boldsymbol{bodySegmentID}$ and $\boldsymbol{oneData}$. In conclusion, each time a filter sends a data sample, $\boldsymbol{S_{s,n}}$, in reality it is sending an array of size 17 ($\boldsymbol{segments :: count = sensor :: count}$) with each body segment name, $\boldsymbol{bodySegmentID}$, and its quaternion, $\boldsymbol{oneData}$. This sample is then added to an ADTF timestamp, $\boldsymbol{\lambda_t}$, to be used synchronously. Following the SDK interaction of each system (DataStream SDK and MT SDK) as shown in algorithms 2.1.1 and 2.1.3, it is possible to implement a filter for MVN Awinda sensor data (Algorithm 3.1.2) and Vicon kinematic fit segment data (Algorithm 3.1.3).

Algorithm 3.1.2 RB_TOP80_MVNAWINDA

1: **Init(StageFirst):**
2: $(tMotionSensor)MVNAwinda \leftarrow output$
3: **Init(StageNormal):**
4: **for** each i in $BodySensors$ **do**
5: $BodySensors[i].segmentID \leftarrow \Gamma$
6: **end for**
7: **_Filter::Start():**
8: $Connection$ **in Alg. 2.1.1**
9: $Thread.start()$
10: **Thread:**
11: **for** each i in $BodySensors$ **do**
12: $BodySensors[i].oneData \leftarrow packet.orientationQuaternion()$ **Data in Alg. 2.1.1**
13: **end for**
14: $MVNAwinda \leftarrow BodySensors$

Algorithm 3.1.3 RB_TOP80_VICONNEXUS

1: **Init(StageFirst):**
2: $(tMotionSegment)ViconNexus \leftarrow output$
3: **Init(StageNormal):**
4: **for** each s in $BodySegments$ **do**
5: $BodySegments[i].segmentID \leftarrow \Gamma$
6: **end for**
7: **_Filter::Start():**
8: $Connection$ **in Alg. 2.1.3**
9: $Thread.start()$
10: **Thread:**
11: **for** each i in $BodySegments$ **do**
12: $BodySegments[i].oneData \leftarrow MyClient.GetSegmentGlobalRotationQuaternion()$ **Data in Alg. 2.1.3**
13: **end for**
14: $ViconNexus \leftarrow BodySegments$

Each filter is then used at a higher level in ADTF with the purpose of recording synchronized data to the hard drive as illustrated in Figure 3.3.

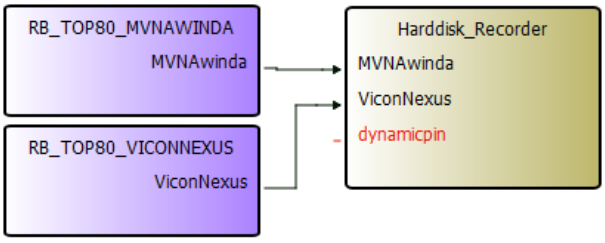


Figure 3.3: ADTF pipeline for DSE evaluation recording.

Evaluation (MATLAB)

MATLAB was used for the second part, loading each recording through the dat2mat script (Figure 2.21) and post-processing them (Figure 3.4). This part of the toolchain has three main functions:

1. Data synchronization;

2. Graphical representation;

3. Metrics generation.

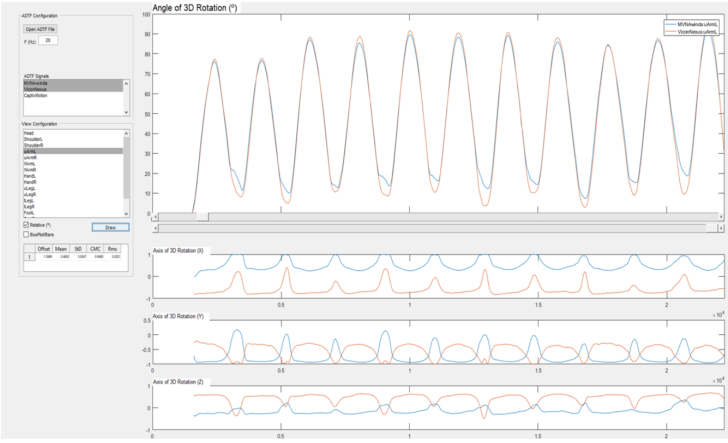


Figure 3.4: MATLAB sub-toolchain for DSE evaluation.

Using the definition of equation 3.5, the first function uses Algorithm 3.1.4 to create new timestamps, $\boldsymbol{\Lambda_{t'}}$, from a custom user-defined frequency, $\boldsymbol{\Gamma_f}$, and interpolate consecutively, $\boldsymbol{n}$, all samples of each quaternion, $\boldsymbol{Q_{s,n,i}}$, from each segment, $\boldsymbol{i}$, from each system, $\boldsymbol{s}$, considering its real timestamp, $\boldsymbol{\lambda_t}$. This way new samples are created for each system but with identical timestamps, $\boldsymbol{\delta_{s,t',i}}$, generating synchronized data samples for all variables $\boldsymbol{B_{s,n,i} \rightarrow \upsilon_{s,t',i}}$, $\boldsymbol{Q_{s,n,i} \rightarrow \varphi_{s,t',i}}$. Note that interpolation needs to be related with the type of data being interpolated, therefore using Spherical Linear Quaterninon Interpolation (SLERP) for the quaternion data.

$$
\begin{aligned}
S_{s,n,i} &= \{bodySegmentID_{s,n,i}, oneData_{s,n,i}\} \\
&\equiv \\
S_{s,n,i} &= \{B_{s,n,i}, Q_{s,n,i}\} \\
\delta_{s,t',i} &= \{\upsilon_{s,t',i}, \varphi_{s,t',i}\}
\end{aligned}
\tag{3.5}
$$

Algorithm 3.1.4 DSE SYNCHRONIZATION

$T_{ini} \leftarrow \begin{cases} \lambda_t, S_{1,0} \wedge S_{2,n} \neq \emptyset \\ \lambda_t, S_{2,0} \wedge S_{1,n} \neq \emptyset \end{cases}$
$T_{end} \leftarrow \begin{cases} \lambda_t, S_{1,N} \wedge S_{2,n} \neq \emptyset \\ \lambda_t, S_{2,N} \wedge S_{1,n} \neq \emptyset \end{cases}$
$T_{range} \leftarrow T_{end} - T_{ini}$
$T \leftarrow \frac{T_{range}}{1/f}$
$\Lambda_{t'} \leftarrow 0, \frac{1}{\Gamma f}, ..., T_{range}, t' = 0, .., T$
$\delta_{s,t',i}, t' = 0, .., T$
for each s in $\delta_{s,t',i}$ **do**
  for each t' in $\Lambda_{t'}$ **do**
    $t_{ini} \leftarrow \lambda_t \cap S_{s,n}, t < t'$
    $t_{end} \leftarrow \lambda_t \cap S_{s,n}, t > t'$
    $ratio \leftarrow \frac{t'-t_{ini}}{t_{end}-t_{ini}}$
    for each i in $\delta_{s,t',i}$ **do**
      $q_{ini} \leftarrow Q_{s,n,i}, t < t'$
      $q_{end} \leftarrow Q_{s,n,i}, t > t'$
      $\varphi_{s,t',i} \leftarrow SLERP(q_{ini}, q_{end}, ratio)$
    end for
  end for
end for

The second function gives the graphical representation of $\boldsymbol{\theta_t}$ in equation 3.4 of each selected body segment (Table 3.1) and system (Figure 3.2). In this case, $\boldsymbol{q_t}$ represents the quaternion of a segment from a system, $\boldsymbol{\varphi_{s,t',i}}$, across the entire timeline, $\boldsymbol{\Lambda_{t'}}$. The third and final function calculates the error for each segment, $\boldsymbol{\xi_{t',i}}$, through equation 3.6. The axis-angle from both systems $\boldsymbol{\theta_{1,t'}}$ and $\boldsymbol{\theta_{2,t'}}$ is used to calculate the dynamic error of the segment across the timeline, $\boldsymbol{\mu_i}$,$\boldsymbol{\sigma_i}$.

$$\begin{aligned} \theta_{1,t',i} &= Eq\ 3.4(q_t \leftarrow \varphi_{1,t' \rightarrow T,i}) \\ \theta_{2,t',i} &= Eq\ 3.4(q_t \leftarrow \varphi_{2,t' \rightarrow T,i}) \\ \xi_{t',i} &= |\theta_{1,t',i} - \theta_{2,t',i}| \\ \therefore \mu_i &= \frac{\sum_{t'=0}^{T} \xi_{t',i}}{T} \\ \therefore \sigma_i &= \sqrt{\frac{\sum_{t'=0}^{T} (\xi_{t',i} - \mu_i)^2}{T-1}} \end{aligned} \tag{3.6}$$

3.1.2.2 Dynamic full body evaluation

Dynamic full body evaluation was focused on understanding how the ISB model behaves against the Vicon kinematic fit model. To accomplish it, both systems need to be recorded synchronously and each joint referenced. Comparison is based on the Euclidian error, $\boldsymbol{\xi_{t',j}}$, between joints, $\boldsymbol{j}$, (equation 3.2) where

$\boldsymbol{Pxyz_{1,j}}$ represents the cartesian position for each of the 20 joints from the first system, and $\boldsymbol{Pxyz_{2,j}}$ from the second. As it was mentioned before, the kinematic fit model does not match completely the ISB model, therefore there is the need to relate each joint of both systems (Table 3.2). Since each model is in their respective world coordinate system, the models need first to be spatially aligned.

Table 3.2: ISB joints reference with kinematic fit segment origin (joint).

$S_{s,n}$	**Kinematic Fit**	**ISB**	$S_{s,n}$	**Kinematic Fit**	**ISB**
Head	Head_End	Head	LeftLowerLeg	L_Tibia	Left Lower Leg
LeftUpperArm	L_Elbow	Left Upper Arm	RightLowerLeg	R_Tibia	Right Lower Leg
RightUpperArm	R_Elbow	Right Upper Arm	LeftFoot	L_Foot	Left Foot
LeftForearm	L_Wrist	Left Forearm	RightFoot	R_Foot	Right Foot
RightForearm	R_Wrist	Right Forearm	LeftToe	L_Toe	Left Toe
LeftHand	L_Wrist_End	Left Hand	RightToe	R_Toe	Right Toe
RightHand	R_Wrist_End	Right Hand	Neck	Head	Neck
LeftUpperLeg	L_Femur	Left Upper Leg	Chest	R_Collar	T8
RightUpperLeg	R_Femur	Right Upper Leg	Pelvis	LowerBack	Pelvis

Similar to the previous evaluation, a toolchain was developed consisting of two sub-toolchains: (1) synchronized recording through ADTF and (2) graphical quantitative evaluation through MATLAB.

Recording (ADTF)

The first sub-toolchain requires the development of two distinct filters for data capture of MVN Awinda ISB model, and Vicon kinematic fit model. Joint data referencing is done as a filter initialization (Table 3.2), although it is possible to customize the information in ADTF before boot, $\boldsymbol{\Gamma}$. With this information, each filter is able to use the same output data structure $\boldsymbol{BodyJoints}$ and $\boldsymbol{BodySegments}$ (Figure 3.5).

```
//structure that contains all the data related to each joint
typedef struct tMotionJoint {
    tChar bodyjointID[15];      //body joint name
    tFloat64 XYZData[3];        //cartesian sample (x,y,z)
    tFloat64 QuatData[4];       //quaternion sample (x,y,z,w)
}tMotionJoint;
//structure that will allow for the creation of a buffer of synced data to be sent to ADTF
typedef struct tMotionBodyJoints {
    tMotionJoint BodyJoints[joints::count];
}tMotionBodyJoints;
```

A

```
//structure that contains all the data related to each segment
typedef struct tMotionSegment {
    tChar bodySegmentID[15];    //body segment name
    tFloat64 XYZData[3];        //cartesian sample (x,y,z)
    tFloat64 QuatData[4];       //quaternion sample (x,y,z,w)
}tMotionSegment;
//structure that will allow for the creation of a buffer of synced data to be sent to ADTF
typedef struct tMotionBodySegments {
    tMotionSegment BodySegments[segments::count];
}tMotionBodySegments;
```

B

Figure 3.5: Filters output for DFE: (A) MVN Awinda, (B) Vicon Nexus.

Although the structure data type is different ($\boldsymbol{tMotionJoint \neq tMotionSegment}$), the relevant data format is similar $\boldsymbol{XYZData}$ and $\boldsymbol{QuatData}$. Following the SDK interaction of each system (DataStream SDK and User Datagram Protocol (UDP) SDK) as shown in algorithms 2.1.3 and 2.1.2, it is possible

to implement a filter for MVN Awinda simplified ISB model joint data (Algorithm 3.1.5) and Vicon kinematic fit segment data (Algorithm 3.1.6). Each filter is then used at a higher level in ADTF with the purpose of recording synchronized data to the hard drive as illustrated in Figure 3.6.

Algorithm 3.1.5 RB_TOP80_MvnAwindaFullBody

Init(StageFirst):
$(tMotionJoint)MVNAwindaFullBody \leftarrow output$
Init(StageNormal):
for each j in $BodyJoints$ **do**
  $BodyJoints[j].bodyjointID \leftarrow \Gamma$
end for
_Filter::Start():
  $Connection$ **in Alg. 2.1.2**
  $Thread.start()$
Thread:
for each j in $BodyJoints$ **do**
  $XYZDataByteIndex \leftarrow 28 + j \times 32$
  $QuatDataByteIndex \leftarrow 40 + j \times 32$
  $BodyJoints[j].XYZData \leftarrow DecodeDataToSegmentTranslation(com(XYZDataByteIndex))$ **Data in Alg. 2.1.2**
  $BodyJoints[j].QuatData \leftarrow DecodeDataToSegmentOrientation(com(QuatDataByteIndex))$ **Data in Alg. 2.1.2**
end for
$MVNAwindaFullBody \leftarrow BodyJoints$

Algorithm 3.1.6 RB_TOP80_ViconNexusFullBody

Init(StageFirst):
$(tMotionSegment)ViconNexusFullBody \leftarrow output$
Init(StageNormal):
for each bs in $BodySegments$ **do**
  $BodySegments[bs].segmentID \leftarrow \Gamma$
end for
_Filter::Start():
  $Connection$ **in Alg. 2.1.3**
  $Thread.start()$
Thread:
for each bs in $BodySegments$ **do**
  $BodySegments[bs].XYZData \leftarrow MyClient.GetSegmentGlobalTranslation()$ **Data in Alg. 2.1.3**
  $BodySegments[bs].QuatData \leftarrow MyClient.GetSegmentGlobalRotationQuaternion()$ **Data in Alg. 2.1.3**
end for
$ViconNexusFullBody \leftarrow BodySegments$

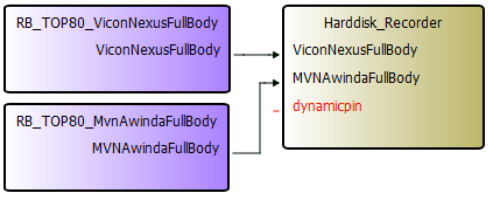


Figure 3.6: ADTF pipeline for DFE evaluation recording.

Evaluation (MATLAB)

A MATLAB sub-toolchain imports and post-processes the recording (Figure 3.4) through four main functions:

1. Data synchronization;
2. Data spatial alignment;
3. Graphical representation;
4. Metrics generation.

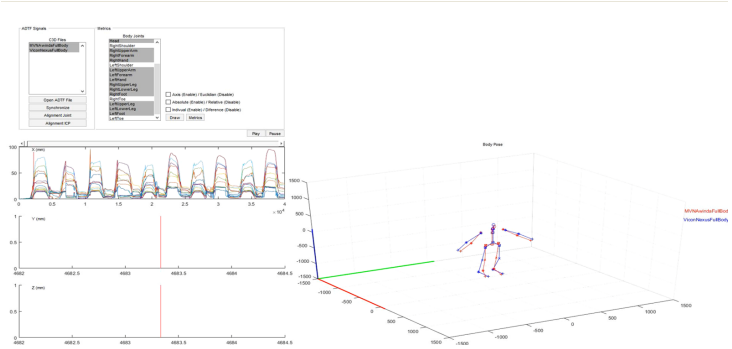


Figure 3.7: MATLAB sub-toolchain for DFE evaluation.

Using the definition in equation 3.7 the first function is similar to Algorithm 3.1.4 (i.e. new synchronized samples are created from the raw ADTF ones, through interpolation), although the main difference comes in the interpolation method. In the present case, linear interpolation is used for joints' position $\boldsymbol{k_{s,n,j} \rightarrow \kappa_{s,t',j}}$, and SLERP is used for the orientation data $\boldsymbol{Q_{s,n,j} \rightarrow \varphi_{s,t',j}}$.

$$
\begin{aligned}
S_{s,n,j} &= \{bodySegmentID_{s,n,j}, XYZData_{s,n,j}, QuatData_{s,n,j}\} \\
&\equiv \\
S_{s,n,j} &= \{B_{s,n,j}, k_{s,n,j}, Q_{s,n,j}\} \\
\delta_{s,t',j} &= \{\upsilon_{s,t',j}, \kappa_{s,t',j}, \varphi_{s,t',j}\}
\end{aligned}
\tag{3.7}
$$

The second function (Figure 3.8) aligns both models through a two-step process: first, a coarse alignment is performed using equation 3.3 to reference each joint $\boldsymbol{j}$ with respect to the pelvis one $\boldsymbol{T_j^{pelvis}}$; then, the Finite Iterative Closest Point (FICP) alignment [98] Algorithm 3.1.7 is used to better align both models. In the latter, each full body sample $\boldsymbol{\kappa_{s,t',j}}$ becomes a 3D mesh, $\boldsymbol{M_{s,t'}}$, giving each joint a vertex xyz position. The

latter step allows to minimize the differences between models while distributing the initial coarse alignment, $T_{\delta_{1,t',j}}^{\delta_{2,t',j}}$, error through the entire body.

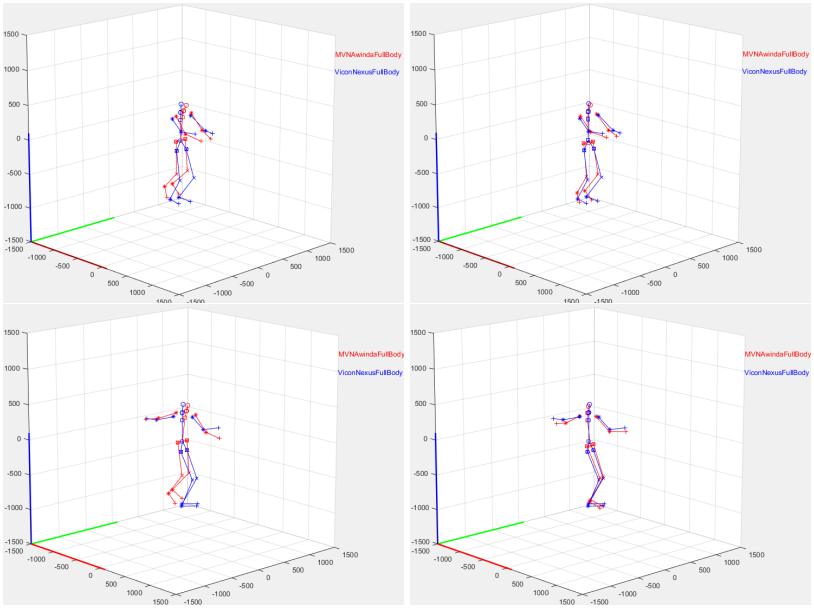


Figure 3.8: DFE MATLAB spatial alingment: 1^{st} column represents initial alignment with equation 3.3, and 2^{nd} column represents the second step of alignment with FICP.

Algorithm 3.1.7 DFE ICP

1: $M_{s,t'} \leftarrow \kappa_{s,t',j}$
2: **for** each t' in $M_{s,t'}$ **do**
3: $\quad T_{\delta_{1,t',j}}^{\delta_{2,t',j}} \leftarrow FICP(M_{s,t'})$
4: $\quad$ **for** each j in $\delta_{s,t',j}$ **do**
5: $\quad\quad \kappa_{2,t',j} \leftarrow T_{\delta_{1,t',j}}^{\delta_{2,t',j}} \cdot \kappa_{2,t',j}$
6: $\quad$ **end for**
7: **end for**

The third function gives the graphical representation of each body model, $\delta_{s,t',j}$, joint cartesian position, $\kappa_{s,t',j}$, (Table 3.2) across the recording timeline, $t' \rightarrow T$, as well as the Euclidian error, $\xi_{t',j}$, between joints (Figure 3.7).

The forth and final function calculates all joints' errors between systems μ_j,σ_j as well as the error graphical representation, $\xi_{t',j}$, (equation 3.8) for the third part.

$$
\begin{aligned}
Pxyz_{s,t',j} &= \kappa_{s,t',j} - \kappa_{s,0,j} \\
\xi_{t',j} &= Eq\ 3.2(Pxyz_{s,j} \leftarrow Pxyz_{s,t' \rightarrow T,j}) \\
\therefore \mu_j &= \frac{\sum_{t'=0}^{T} \xi_{t',j}}{T} \\
\therefore \sigma_j &= \sqrt{\frac{\sum_{t'=0}^{T} (\xi_{t',j} - \mu_j)^2}{T-1}}
\end{aligned}
\tag{3.8}
$$

3.2 Evaluation results

In this section, the inertial suit evaluation is described. As it was mentioned in Section 3.1, there are four evaluation procedures, SSE, SFE, DSE and DFE. For each procedure, several toolchains were developed and with them several evaluation recordings were perfomed. All MVN Awinda sensors were updated to V4.2.1, and motion capture was performed with MVN Studio 4.97.1.

3.2.1 Static sensor evaluation

To understand the sensors' behaviour at the lowest level, each axis was evaluated as illustrated in Figure 3.9. A full 360^o rotation with 11.25^o increments, $\boldsymbol{degree}$, for each axis was performed. At each incremental rotation, 300 samples, $\boldsymbol{nS}$, were recorded for 5 seconds.

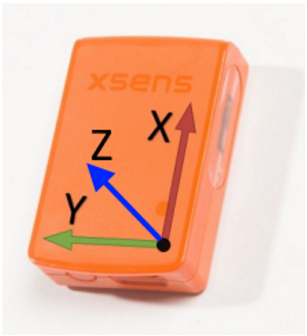


Figure 3.9: Sensor axes and associated rotations: X - roll, Y - pitch and Z - yaw. Adapted from MVN Users' Manual [19].

In Figure 3.10, it is possible to understand that the sensor is very accurate for the Roll and Pitch rotations, but it showed higher errors for Yaw, probably due to the fact that it is highly sensitive to local distortions of the Earth's magnetic field.

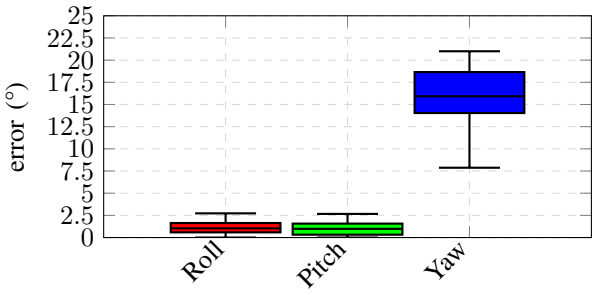


Figure 3.10: Sensor's angular error in all axes.

3.2.2 Static full body evaluation

To understand the full body static behaviour, three evaluation procedures were performed (Figure 3.11):

- EV1: Closed circuit driven during 80 seconds for 500 meters;
- EV2: Closed circuit driven during 120 seconds for 500 meters;
- EV3: City to city circuit driven during 2400 seconds for 40 kilometers.

The purpose was to compare each final N-Pose (after exiting the car) with the first one (before entering the car), and from that evaluate the Euclidian error of each joint.

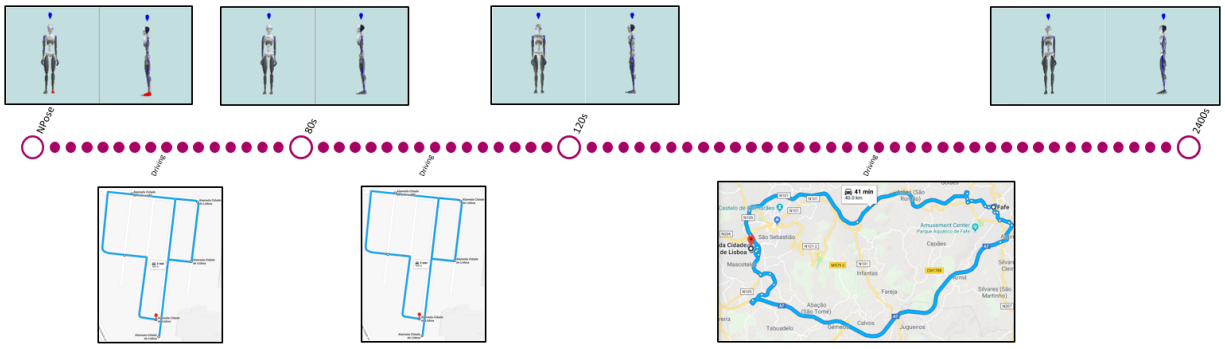


Figure 3.11: Evaluation procedures in regard to circuit, distance, time and full body pose qualitative result.

In Figure 3.12, it is possible to understand that although there is an increased error related with driving time (magnetic sensitivity, sensor/strap displacement, inherent drift) and joint distance from the pelvis (error associated with kinematic forwarding), the inertial suit output is quite stable. If drift was the only stability factor, worst case would be: $EV1 = 750\ \mu m/s$, $EV2 = 416\ \mu m/s$ and $EV3 = 39.5\ \mu m/s$, where it is possible to see that drift is not constant throughout the evaluation procedures. This observation shows that error acumulation has limited relation with time.

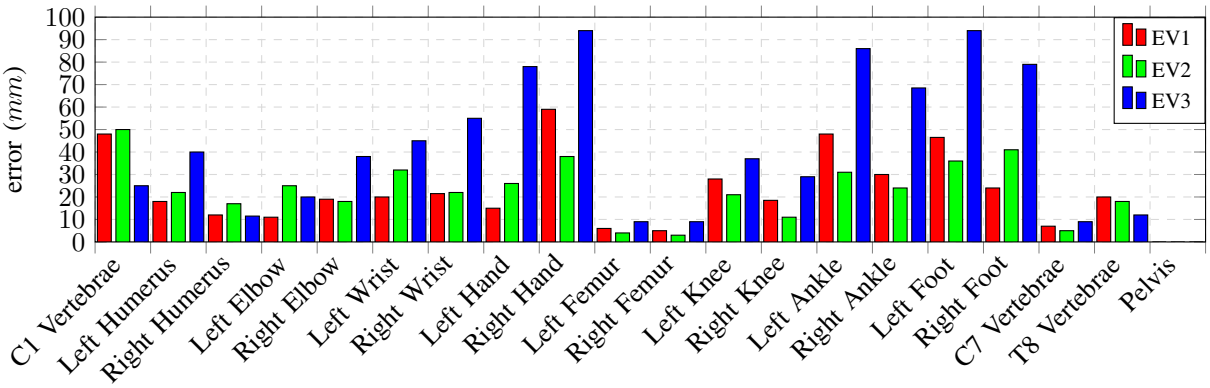


Figure 3.12: Joints's (Table 2.1) Euclidian error, $\boldsymbol{\xi_j}$, for each SFE evaluation procedure (EV1, EV2 and EV3).

3.2.3 Dynamic sensor evaluation

To understand how the sensors behave in the human body, we placed them as indicated by the supplier (Figure 2.1) and evaluated them at a lower level. This concept allowed us to understand how each sensor behaves in each segment, in terms of rotation range. To contextualize this understanding with body movements, a total of three recording procedures were established. These had the purpose of evaluating a specific body movement plane: coronal (RC), sagittal (RS) and transversal (RT). For each one, a full body motion procedure was performed and periodically repeated for ten cycles. Each procedure was recorded five times to have suficient statistical data.

Segment results in coronal and sagittal planes show the worst results (Figures 3.13A and 3.13B) in terms of segments with higher soft tissue (upper arms and legs), given the increased frequency of muscular contractions. Feet also presented sub-optimal results, but are more related to the sensor's proximity to magnetic distortions from the iron foundations of the building. Transversal plane (Figure 3.13C) show the best results with a segment median error, $\boldsymbol{\mu_i}$, lower than 10^o, which is related to the fact that it required less soft tissue contractions. Full body segment evaluations show a median full body error, $\boldsymbol{\mu_i}$, lower than 5^o across planes (Figure 3.13D).

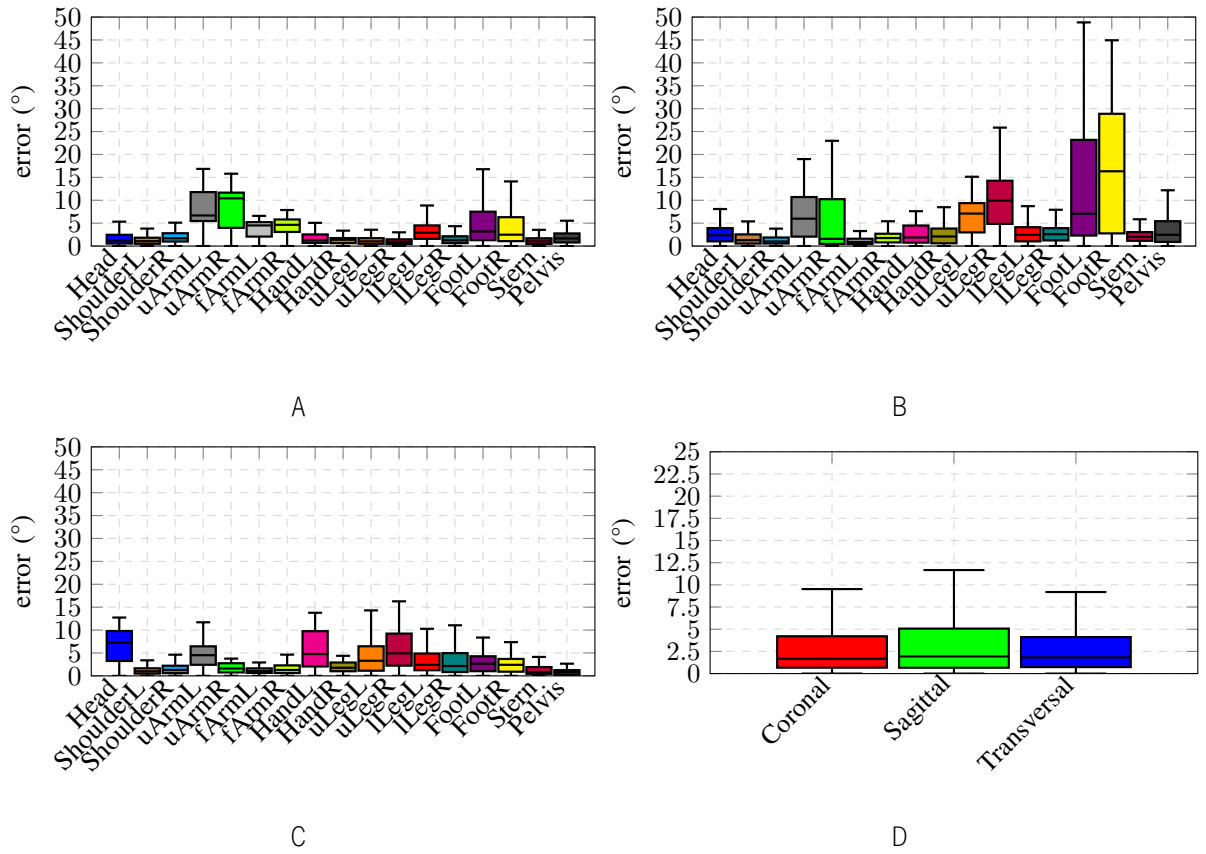


Figure 3.13: Segments' angular error. (A) coronal plane, (B) sagittal plane, (C) transversal plane and (D) full body in all planes.

Due to the higher error found in the upper arms and legs, an extra segment evaluation was performed. In this case, new sensor positions were defined and evaluated (Table 3.3), and the results can be seen in Figures 3.14A and 3.14B. The results show that the best position for fixation is the $Greater\ trochanter$ for the upper leg, and $Triceps\ lateral$ for the upper arm.

Table 3.3: Upper Arm and Upper Leg segment evaluations.

Evaluation	Sensor Leg Position	Sensor Arm Position	Movement Plane
P1C	Greater trochanter	Triceps lateral	Coronal
P1S	Greater trochanter	Triceps lateral	Sagital
P2C	Iliotibial tract	Triceps long	Coronal
P2S	Iliotibial tract	Triceps long	Sagital
P3C	Bicepts femoris	Triceps medial	Coronal
P3S	Bicepts femoris	Triceps medial	Sagital
P4C	Semitendinosus	Biceps	Coronal
P4S	Semitendinosus	Biceps	Sagital

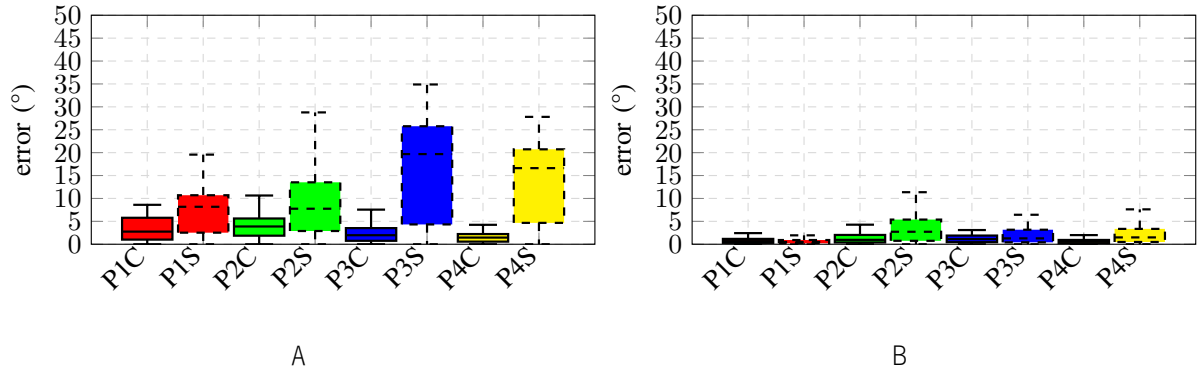


Figure 3.14: Upper arm and leg orientation error in coronal and sagittal planes for different positions, p (from 1 to 4). (A) Upper Arm and (B) Upper Leg.

3.2.4 Dynamic full body evaluation

To understand how the joints' position behave in the human body, we evaluated the complete output from the inertial suit (Figure 2.1). This evaluation gives a better understanding of how the Awinda motion capture system behaves against the Vicon motion capture system in terms of joints' position. To contextualize this understanding with body movements, the same three recording procedures from the segment evaluation were used. An extra 4^{th} procedure was added, where a driving simulation (outside the car) was performed and assessed (RDr).

In terms of joints' evaluations, the transversal plane maintains the best results (Figure 3.15C). Coronal and sagittal planes show worst errors (Figures 3.15A and 3.15B), specifically for the lower body extremities in the coronal plane. This is due to error propagation from the kinematic forwarding method, and the fact that the coronal plane movements had considerably higher feet joint movement range compared with the sagittal and transversal planes. In regard to the 4^{th} procedure, the error increased (Figure 3.15D), possibly due to the increase in range of motion in almost every joint, specifically the right hand (simulating its movement to the back part of the car seat). Full body joint results in Figure 3.15E show a median error, $\boldsymbol{\mu_j}$, lower than $15mm$ for the coronal, sagittal and transversal planes, and $30mm$ for the driving simulation.

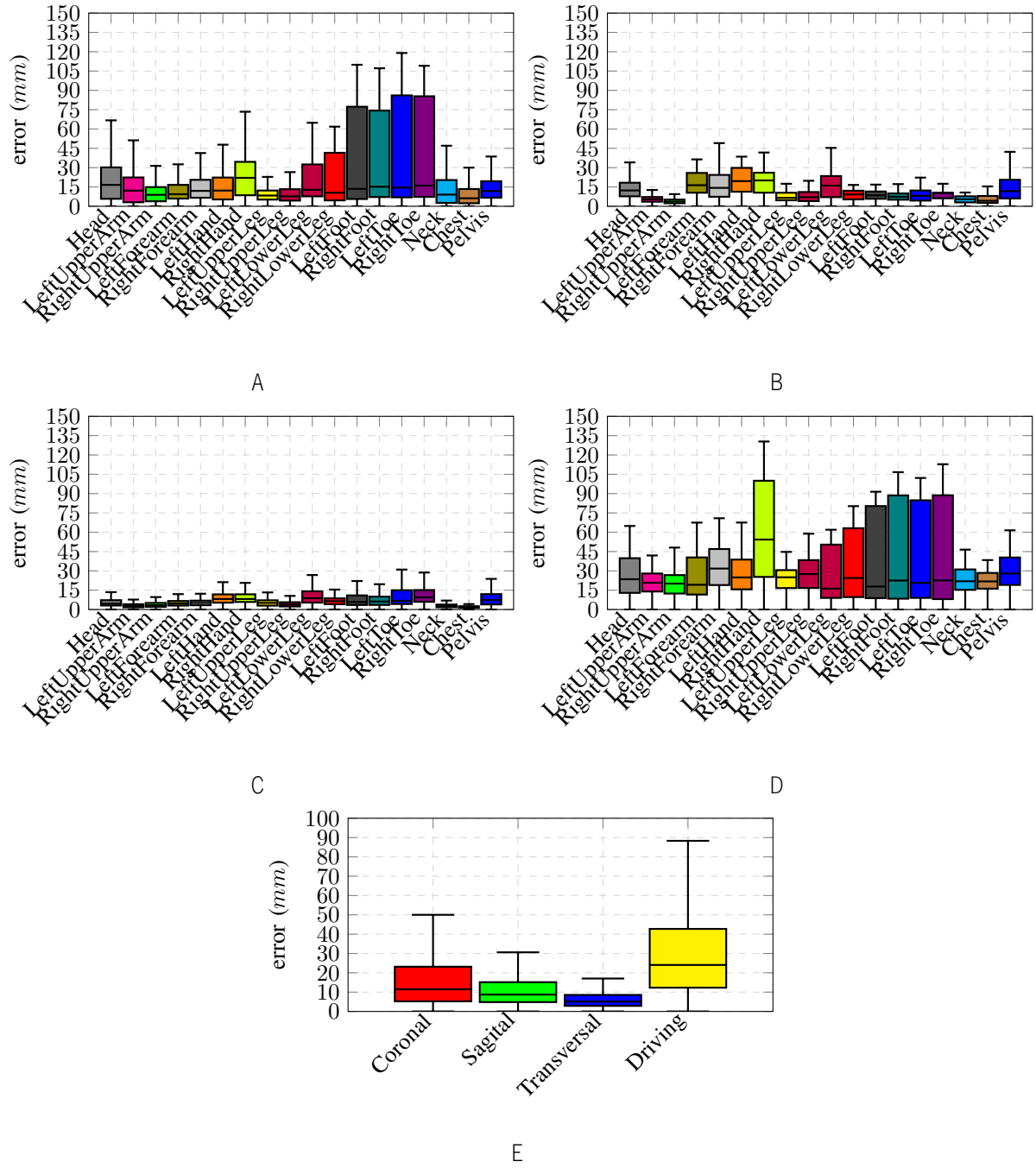


Figure 3.15: Joints' positional error. (A) coronal plane, (B) sagittal plane, (C) transversal plane, (D) driving simulation and (E) full body in all evaluations.

3.3 Discussion and Conclusions

We evaluated a specific inertial suit, highlighting its limitations through an extensive evaluation procedure that separates itself from other more specific methods [55] where the evaluation is focused mainly in the calibration procedure and the full body kinematics output. Although there was no focus in evaluating the calibration procedure, a specific sub-toolchain was developed that allowed us to learn the best stance for

the human subject in the calibration step. This sub-toolchain is not used each time we perform a recording due to the fact that it requires both the Awinda and the Vicon full body suits, creating to much entropy while recording. Sub-optimal performance was detected at a sensor level for the Yaw rotation (Figure 3.10), however this behaviour did not propagate to the full body analysis. Full body analysis showed larger errors for sensors in segments with increased soft tissue, or performing an higher Range of Motion (RoM) (Figures 3.13A and 3.13B). This source of error propagates into the joints' position analysis, due to its inherited kinematic forwarding pipeline, showing the larger errors in the furthest body joints wrt. to the pelvis joint (Figures 3.15A, 3.15B and 3.15D). We believe that this evaluation to the inertial suit brings a better understanding on its behaviour and limitations.

Chapter 4

Real Dataset Toolchain

The present chapter follows up on the previous one, focusing on the real dataset generation problem. Preliminary versions of the work presented in this chapter were presented in the paper **"A system for the generation of in-car human body pose datasets"**, currently in revision process.

In the first section of this chapter, the real dataset toolchain (previously mentioned in Section 1.5.2) development is shown in-depth. The complete real dataset generation pipeline is shown through three main parts: recording in Section 4.1.1, alignment in Section 4.1.2, and rendering in Section 4.1.3. Recording (Section 4.1.1) shows the Automated Data and Time-Triggered Framework (ADTF) sub-toolchains that allow for real time synchronized recording, describing each developed filter. Alignment and rendering (sections 4.1.2 and 4.1.3) are based on a single MATLAB sub-toolchain, shown in-depth in both sections. Specific temporal and spatial calibration methods and algorithms are presented in Section 4.1.2, while dataset specific data rendering is shown in Section 4.1.3.

In the second section of this chapter, the entire toolchain's evaluation and potential interest is shown. Two evaluation procedures were defined, one focused on the real data generation itself, and a second focused on the real data application for human body pose estimation. The first evaluation procedure (Section 4.2.1.1) is divided in tightly secured human motion and free human motion, car seat position and human actions, in order to show how the recorded human body pose behaves in different scenarios. The second evaluation procedure (Section 4.2.2) shows how real datasets generated by the toolchain can improve the accuracy of different Machine Learning (ML) based methods for human body pose detection.

The chapter proceeds with the discussion of the obtained results, and its main conclusions. Results from evaluation procedures show that the toolchain falls behind the market gold standard in terms of out-car scenario, however it improves considerably for the in-car scenario, being able to capture the human body pose wrt. a Time-of-Flight (ToF) sensor. This understanding goes behond the in-car scenario, opening possibilities

for highly occluded scenarios. Toolchain generated data was shown as being valuable for ML based methods, increasing their detection accuracy. Dataset sources of error were identified across chapters 3 and 4 and future error minimization approaches were proposed.

Contents

4.1 Toolchain Overview

In this section, the previously selected ground-truth system (Chapter 3) is used for the purpose of developing a toolchain capable of generating ToF images with associated human body pose ground-truth for in-car scenario. To achieve such toolchain, several systems are required:

- a ToF sensor for image capture ($\boldsymbol{C}$ being the position of the camera's optical center);
- an Inertial suit (namely a MVN Awinda) for relative human body pose ground-truth ($\boldsymbol{A}$ being the head joint, and $\boldsymbol{J}$ each one of the remaining body joints);
- a global object positioning system, such as the Vicon system ($\boldsymbol{W}$ being the Vicon's global coordinate system, and $\boldsymbol{O}$ the subject's head object tracked by it);
- a car testbed.

As Figure 4.1 illustrates, there is the need to indirectly estimate $\boldsymbol{T_J^C}$ in order to project each joint into the ToF perspective. This estimation adds a certain complexity with the added systems, making it necessary to directly estimate all other transformations between systems. With it, there is a need to spatially and temporally align the data, to correctly project the human body pose information into the ToF camera's coordinate system.

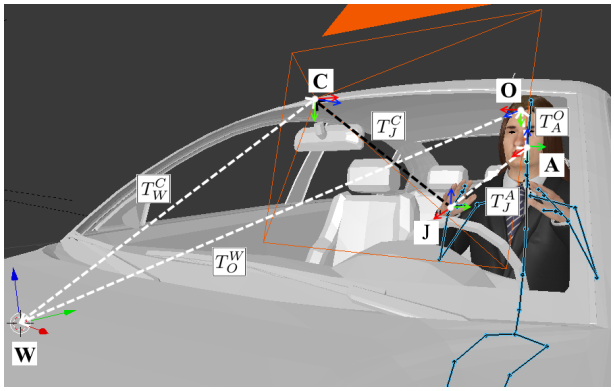


Figure 4.1: 3D representation of coordinate systems when recording a real dataset. $\boldsymbol{W}$: Vicon global coordinate system; $\boldsymbol{C}$: ToF optical center and orientation; $\boldsymbol{O}$: subject's head object tracked by the Vicon system; $\boldsymbol{A}$: inertial suit head joint; $\boldsymbol{J}$: inertial suit joints.

To satisfy the abovementioned requirements, the toolchain implements the pipeline illustrated in Figure 4.2, where the entire real time recording procedure is done with ADTF sub-toolchains, and the alignment and rendering is done in a MATLAB sub-toolchain (Figure 4.3).

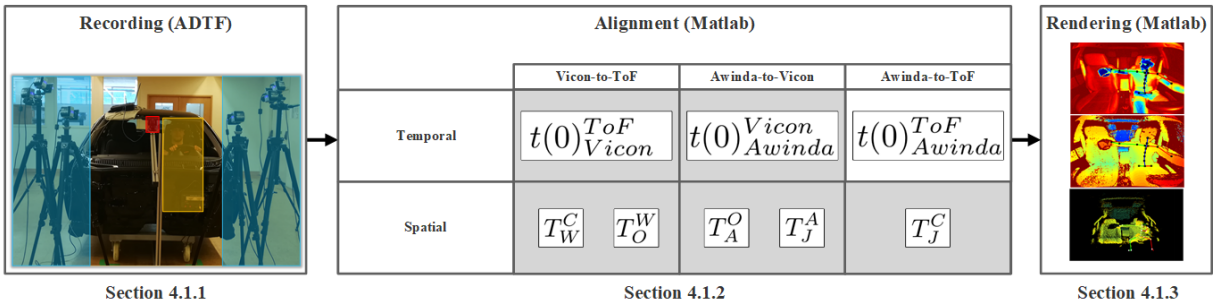


Figure 4.2: Overview of the toolchain pipeline. Recording: blue highlight represents the Vicon system; orange highlight represents the MVN Biomech Awinda system; red highlight represents the ToF camera. Alignment: gray highlights correspond to the developed algorithms, with the other variables being determined by concatenation of the other algorithms' output.

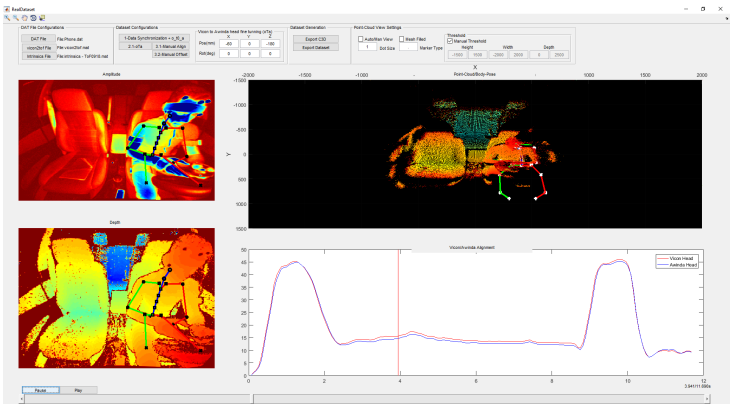


Figure 4.3: MATLAB sub-toolchain for real dataset generation.

4.1.1 Recording (ADTF)

Dataset recording is done for all relevant systems: image data is recorded from a ToF sensor; relative body pose data is recorded from the MVN Awinda inertial suit; global positioning data is recorded with Vicon Nexus through the creation of a virtual head object for the human subject. Note that this object was selected for two main reasons: (1) low soft tissue related errors for both Vicon and inertial suit's markers/sensors; and (2) best joint visibility for the Vicon system in an in-car environment, generating more valid data.

The entire recording process follows the same philosophy as previous toolchains in Chapter 3, where there is the need to use ADTF to implement a sub-toolchain for real time recording. As it was already mentioned, there are three main systems requiring an ADTF filter (Figure 4.4A), shown in subsections: (4.1.1.1) ToF camera, (4.1.1.2) MVN Awinda Full Body, and (4.1.1.3) Vicon head tracker.

Since the temporal alignment between the ToF sensor and the Vicon system is performed independently prior to any acquisition, a separate recording procedure was necessary (Figure 4.4B), being shown in detail in subsections 4.1.1.4 and 4.1.2.1.

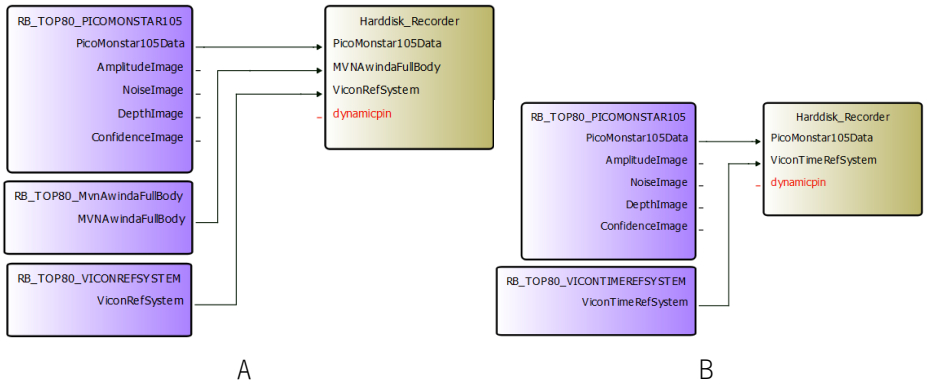


Figure 4.4: ADTF sub-toolchain for real dataset generation: (A) main real dataset recording, and (B) recording for Vicon to ToF temporal alignment.

4.1.1.1 ToF camera

Despite the latest toolchain using one single ToF sensor, being the latest Bosch sensor iteration the Pico Monstar 105, all sensor iterations will be shown as well as their in-depth implementation.

Melexis EVK75023 and EVK75123

Although different evaluation kits, both sensors use the same BltTofApi Software Development Kit (SDK), allowing for an identical filter implementation. In terms of filter output data, independent camera frames and a custom data structure (i.e. comprised by the data necessary for the real dataset [Figure 4.5A]) are used, where $\boldsymbol{Amplitude}$ and $\boldsymbol{PointCloud}$ store the amplitude, depth and point cloud data from the sensor at a 320x240 pixel resolution. Sample data $\boldsymbol{S_{s,n}}$ is comprised by the $\boldsymbol{EVK75123\_Frames}$ structure variable, this sample is then added to an ADTF timestamp $\boldsymbol{\lambda_t}$ to be used synchronously. One of the main differences between the ToF filter and previous ones in Chapter 3 is the ability to configure the sensor itself. This is done through the initialization process in ADTF, where user data, $\boldsymbol{\Gamma}$, (Figure 4.5B) is read by the filter and sent to the sensor.

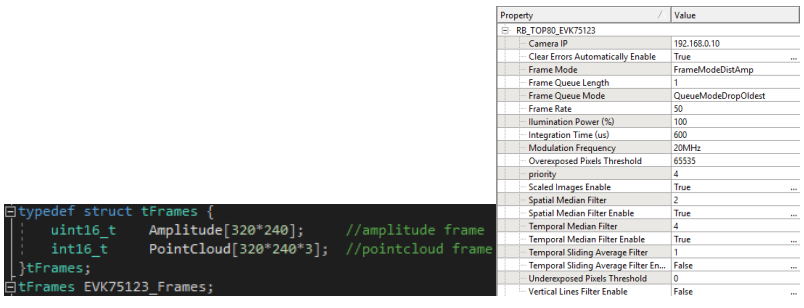


Figure 4.5: EVK75123 filter: (A) output, and (B) user configurations, $\boldsymbol{\Gamma}$.

However, in order for the filter to send user data to the sensor, it is important to understand the SDK interaction. The evaluation kit has three main memory areas: (1) one for global registers where global configurations such as image filters are defined (Figure 4.6), (2) and (3) are considered memory tables $\boldsymbol{T1}$ and $\boldsymbol{T2}$ for the purpose of using two diferent sensor configurations in mixed mode (more range with better Signal-to-Noise Ratio (SNR) at close range).

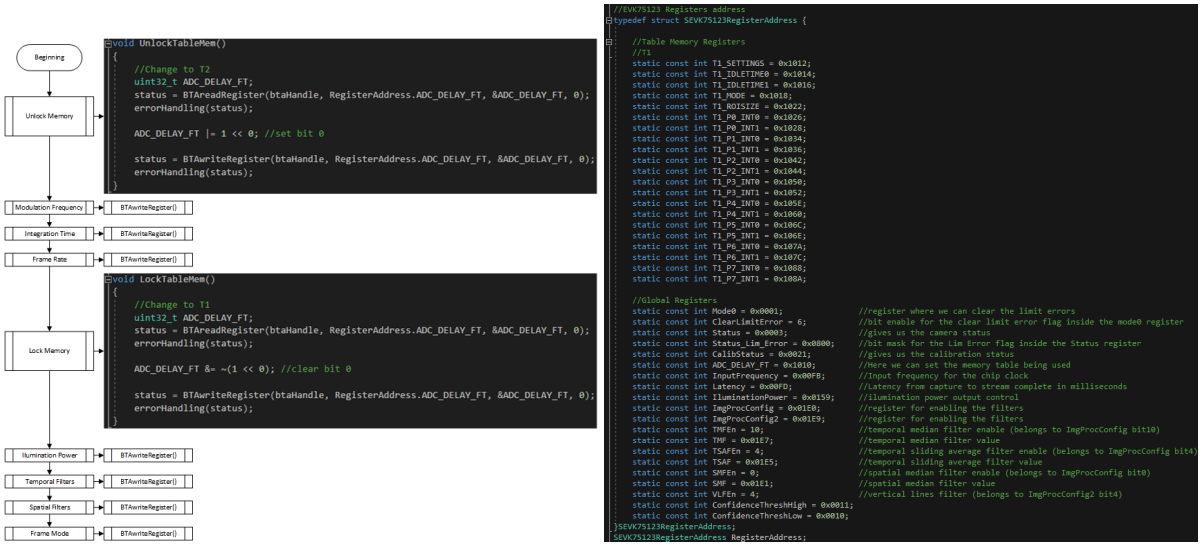
Addr (hex)	Register Name	Default Value (hex)	R/W	Description
01E0	ImgProcConfig	28C0	R/W	Bit[0]: 1...enable Median Filter Bit[3]: 1...enable Bilateral Filter Bit[4]: 1...enable Sliding Average Bit[6]: 1...enable Wiggling compensation Bit[7]: 1...enable FPPN compensation Bit[10]: 1...enable FrameAverage Filter Bit[11]: 1...enable Temperature compensation Bit[13]: 1...enable offset via register DistOffset0

Figure 4.6: One of the EVK75123 global registers, ImgProcConfig manages the enable/disable of embedded filters. Image adapted from [20].

Memory tables $\boldsymbol{T1}$ and $\boldsymbol{T2}$ are mainly used to configure the duty cycle times for the Pulse Modulation (PM), $\boldsymbol{IDLETIME\#, INT\#}$, which are calculated in regard to the modulation frequency, $\boldsymbol{f_m}$, frame rate, $\boldsymbol{f_{in}}$, and integration time, $\boldsymbol{IT}$, number of phases, $\boldsymbol{NPhase}$, and Central Processing Unit (CPU) processing frequency, $\boldsymbol{CPUclock}$, (equation 4.1).

$$
\begin{aligned}
PhaseReadTime &= \frac{1}{\frac{CPUclock}{2}} \cdot \frac{320}{2} \cdot 240 \\
\therefore IDLETIME\# &= (\frac{1}{f_{in}} - NPhase \cdot (IT + PhaseReadTime)) \cdot f_m \\
\therefore INT\# &= IT \cdot f_m
\end{aligned}
\tag{4.1}
$$

Due to this fact, one must implement an entire sensor configuration procedure for the **_Filter::Start()** function (Figure 4.7A), where specific registers are used (Figure 4.7B).



A B

Figure 4.7: EVK75123 initialization: (A) graph, and (B) registers address.

Following the SDK interaction in Algorithm 2.1.4, the filter is able to connect and capture data from the sensor, as well as configure it (Figures 4.5B and 4.7A), with the entire filter higher level implementation being shown in Algorithm 4.1.1 as well as the filter itself in Figure 4.8.

Algorithm 4.1.1 RB_TOP80_EVK75123

1: **Init(StageFirst):**

2: $(tFrames)EVK75123Data \leftarrow output$
3: $(tBitmapFormat)AmplitudeImage \leftarrow output$
4: $(tBitmapFormat)DepthImage \leftarrow output$
5: $(tBitmapFormat)BalanceImage \leftarrow output$
6: **Init(StageNormal):**

7: $EVK75123.userconfig \leftarrow \Gamma$ **in Figure 4.5B**
8: **_Filter::Start():**

9: $Connection$ **in Alg. 2.1.4**
10: $RegisterAddress \leftarrow EVK75123.userconfig$ **in Figure 4.7A**
11: $Thread.start()$
12: **Thread:**

13: $EVK75123\_Frames.Amplitude \leftarrow BTAgetAmplitudes(frame)$ **Data in Alg. 2.1.4**
14: $EVK75123\_Frames.PointCloud \leftarrow BTAgetXYZcoordinates(frame)$ **Data in Alg. 2.1.4**
15: $EVK75123Data \leftarrow EVK75123\_Frames$
16: $AmplitudeImage \leftarrow BTAgetAmplitudes(frame)$ **Data in Alg. 2.1.4**
17: $DepthImage \leftarrow BTAgetDistances(frame)$ **Data in Alg. 2.1.4**
18: $BalanceImage \leftarrow frame.channels.data$ **Data in Alg. 2.1.4**

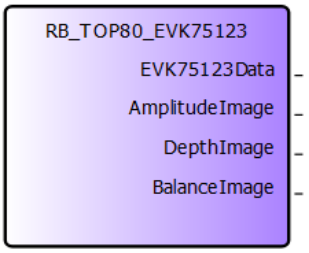


Figure 4.8: ADTF filter for EVK75123.

Gene8 Pico Monstar 105

The final and current ToF sensor in the Bosch roadmap was the Pico Monstar105. This sensor relies on the libroyale SDK [93] for configuration purposes and to capture image data. In terms of filter output data, it uses the same philosophy as the Melexis filters, with independent camera frames and a custom structure comprised by the camera specific real dataset data (Figure 4.9A), where $\boldsymbol{Amplitude}$ and $\boldsymbol{PointCloud}$ preserve the amplitude, depth and point cloud data from the sensor at a 352x287 pixel resolution. Sample data, $\boldsymbol{S_{s,n}}$, is comprised by the $\boldsymbol{PicoMonstar\_Frames}$ structure variable, which is added to an ADTF timestamp, $\boldsymbol{\lambda_t}$, to be used synchronously. During the initialization process, the sensor is configured by reading the ADTF user data, $\boldsymbol{\Gamma}$, (Figure 4.9B) and sending the configuration parameters to the sensor ($\boldsymbol{cameraDevice \rightarrow setUseCase(\Gamma)}$ in Algorithm 2.1.5).

```
typedef struct tFrames {
    tUInt16     Amplitude[xres * yres];     //amplitude frame
    tInt16      PointCloud[xres * yres * 3];    //pointcloud frame
}tFrames;
tFrames PicoMonstar_Frames;
```

A

Property	Value
RB_TOP80_PICOMONSTAR106	
Amplitude Frame Output Enable	True ...
Camera Mode	(0.5-1.5m/15fps/0.6ms)
Confidence Frame Output Enable	True ...
Depth Frame Output Enable	True ...
Noise Frame Output Enable	False ...
PointCloud Output Enable	True ...
priority	1
Scaled Images Enable	False ...

B

Figure 4.9: Pico Monstar 105 filter: (A) output, and (B) user configurations, $\boldsymbol{\Gamma}$.

Through the SDK interaction in Algorithm 2.1.5, the filter is able to connect and capture data from the sensor, with the filter higher level implementation being shown in Algorithm 4.1.2.

Algorithm 4.1.2 RB_TOP80_PICOMONSTAR105

1: **Init(StageFirst):**

2: $(tFrames)PicoMonstar105Data \leftarrow output$
3: $(tBitmapFormat)AmplitudeImage \leftarrow output$
4: $(tBitmapFormat)NoiseImage \leftarrow output$
5: $(tBitmapFormat)DepthImage \leftarrow output$
6: $(tBitmapFormat)ConfidenceImage \leftarrow output$
7: **Init(StageNormal):**

8: $cameraDevice- > setUseCase(\Gamma)$ **in Figure 4.9B**
9: **_Filter::Start():**

10: $Connection$ **in Alg. 2.1.5**
11: $Thread.start()$
12: **Thread:**

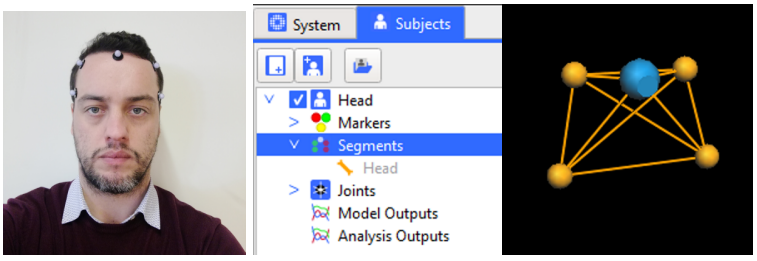
13: $PicoMonstar\_Frames.Amplitude \leftarrow data- > points.grayValue$ **Data in Alg. 2.1.5**
14: $PicoMonstar\_Frames.PointCloud \leftarrow \{data- > points.x, data- > points.y, data- > points.z\}$ **Data in Alg. 2.1.5**
15: $PicoMonstar105Data \leftarrow PicoMonstar\_Frames$
16: $AmplitudeImage \leftarrow data- > points.grayValue$ **Data in Alg. 2.1.5**
17: $NoiseImage \leftarrow data- > points.noise$ **Data in Alg. 2.1.5**
18: $DepthImage \leftarrow data- > points.z$ **Data in Alg. 2.1.5**
19: $ConfidenceImage \leftarrow data- > points.depthConfidence$ **Data in Alg. 2.1.5**

4.1.1.2 MVN Awinda Full Body

For full body pose, there is no need to implement a new filter, with real dataset generation making use of the MVN Awinda filter developed for the DFE evaluation (i.e. Algorithm 3.1.5 in Section 3.1.2.2). This decision comes from the fact that the filter captures the required information for the dataset (full body joints' cartesian position and orientation).

4.1.1.3 Vicon head tracker

To create an head tracker, first an head marker pattern needs to be defined in the subject's head (Figure 4.10A) and then created as an head object in Vicon Nexus (Figures 4.10B and 4.10C).



A B C

Figure 4.10: Head tracker configuration: (A) Head marker pattern, (B) and (C) Vicon Nexus head object creation.

Through the Datastream SDK in Algorithm 2.1.3 and with the information of the head object used as output (Figure 4.11), a specific filter can be created to capture the head segment position and orientation

(Algorithm 4.1.3).

```
typedef struct tMotionObjects {
    tFloat64 HeadRotation[4];       //(x,y,z,w)
    tFloat64 HeadTranslation[3];    //(x,y,z)
}tMotionObjects;
tMotionObjects ViconHead_Data
```

Figure 4.11: Filter output for Vicon head tracker.

Algorithm 4.1.3 RB_TOP80_VICONREFSYSTEM

1: **Init(StageFirst):**
2: $(tMotionObjects)ViconRefSystem \leftarrow output$
3: **Init(StageNormal):**
4: **_Filter::Start():**
5: $Connection$ **in Alg. 2.1.3**
6: $Thread.start()$
7: **Thread:**
8: $ViconHead\_Data.HeadTranslation \leftarrow MyClient.GetSegmentGlobalTranslation()$ **Data in Alg. 2.1.3**
9: $ViconHead\_Data.HeadRotation \leftarrow MyClient.GetSegmentGlobalRotationQuaternion()$ **Data in Alg. 2.1.3**
10: $ViconRefSystem \leftarrow ViconHead\_Data$

4.1.1.4 Vicon marker tracker

In regard to the second recording procedure, to temporally align the ToF camera with the Vicon system, a vicon marker is used as a pendulum while being visible by the ToF and Vicon systems (Figure 4.12).

Figure 4.12: Marker pendulum motion capture through Vicon and ToF sensor, for temporal alignment between systems: blue highlight represents the Vicon marker, and red highlight represents the ToF camera.

This information was recorded with an ADTF sub-pipeline (Figure 4.4B), in order to garantee the temporal latency of the final real dataset recording illustrated in Figure 4.4A. To create such sub-pipeline, there was the need to use the previously created Pico Monstar 105 filter (Algorithm 4.1.2), and the creation of a specific marker recording (Figure 4.13) Vicon filter (Algorithm 4.1.4) using DataStream SDK to capture marker data, $\boldsymbol{Pxyz^{W}_{marker} \equiv ViconTimeRefSystem}$.

```
typedef struct tMotionMarker {
    tFloat64 MarkerTranslation[3];  //(x,y,z)
}tMotionMarker;
tMotionMarker ViconMarker_Data;
```

Figure 4.13: Filter output for Vicon marker tracker.

Algorithm 4.1.4 RB_TOP80_VICONTIMEREFSYSTEM

1: **Init(StageFirst):**
2: $(tMotionMarker)ViconTimeRefSystem \leftarrow output$
3: **Init(StageNormal):**
4: **_Filter::Start():**
5: $Connection$ **in Alg. 2.1.3**
6: $MyClient.EnableMarkerData()$
7: $Thread.start()$
8: **Thread:**
9: $ViconMarker\_Data.MarkerTranslation \leftarrow MyClient.GetMarkerGlobalTranslation()$
10: $ViconTimeRefSystem \leftarrow ViconMarker\_Data$

4.1.2 Alignment (MATLAB)

In order to achieve proper temporal syncronization, a temporal alignment between systems is required. Considering the ToF camera as master to avoid image blur from interpolations, the other two systems have to be synchronized and interpolated to each master timestamp (subsections 4.1.2.1 and 4.1.2.2). This procedure defers from the one used in evaluations DSE and DFE (Sections 3.1.2.1 and 3.1.2.2) where each system was synchronized to new timestamps, thus requiring a new synchronized method, shown in Section 4.1.2.3.

Considering the goal of generating a human body pose ground-truth projected into the ToF coordinate system, several transformations need to be performed, each requiring a calibration procedure. There is the need to spatially align the Vicon system with the ToF camera (subsection 4.1.2.4), as well as the Awinda suit with the Vicon system (subsection 4.1.2.5).

4.1.2.1 Vicon-to-ToF temporal alignment

With the recorded information (Figure 4.4B), for each timestamp two virtual markers were created in the camera's perspective, with $\boldsymbol{xy}$ coordinates: (1) one in the $\boldsymbol{amp_{x,y}}$ frame (i.e. the marker pendulum $\boldsymbol{xy}$ position in the image frame, $\boldsymbol{Pxy_C}$ detected through equation 4.2 due to its saturated IR reflectivity, given by the $\boldsymbol{xy}$ pixel position of the maximum reflectivity pixel), and (2) the 3D→2D projection of the Vicon's marker 3D position in the camera's image frame $\boldsymbol{Pxyz^C_{marker} \rightarrow Pxy^C_{marker}}$ (through equation 4.3).

$$Pxy_C = \underset{x,y}{argmax}(amp_{x,y}) \tag{4.2}$$

$$
\begin{aligned}
Pxyz^{C}_{marker} &= T^{C}_{W} \cdot Pxyz^{W}_{marker} \\
\therefore Pxy^{C}_{marker} &= Pxyz^{C}_{marker} \cdot IM_{C}
\end{aligned}
\tag{4.3}
$$

Equation 4.3 relies on two calibrations: (1) the camera's pre-calibrated intrinsic matrix $\boldsymbol{IM_C}$ (equation 4.4), with $\boldsymbol{f_x}$, $\boldsymbol{f_y}$ being the focal length in pixels, $\boldsymbol{c_x}$, $\boldsymbol{c_y}$ being the optical center in pixels, and $\boldsymbol{sk}$ being the camera axes skew angle; and (2) the Vicon-to-ToF spatial calibration $\boldsymbol{T^C_W}$ (subsection 4.1.2.4).

$$
IM_{C} = \begin{bmatrix} f_x & 0 & 0 \\ sk & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}
\tag{4.4}
$$

Both virtual markers must exist for each timestamp, generating an oscilating wave in the ToF x-axis (Figure 4.14). Assuming that both signals are delayed copys of each other, this projection allows to estimate the time delay $\boldsymbol{t(0)^{ToF}_{Vicon}}$ between both systems through the cross-correlation of the two discrete-time sequences, $\boldsymbol{Pxy_C}$ and $\boldsymbol{Pxy^C_{marker}}$.

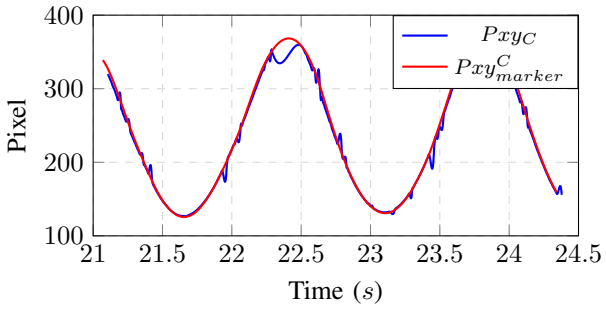


Figure 4.14: Projection of the Vicon's marker in the ToF x-axis (red) and the marker's x-axis coordinate in the image frame (blue), giving the delay between systems $\boldsymbol{t(0)^{ToF}_{Vicon}}$. Spikes in blue line are related to ToF marker centroid perspective parallax error.

4.1.2.2 Awinda-to-Vicon temporal alignment

To determine the temporal mismatch between the Vicon system and the Awinda suit, both systems' head joint quaternions were converted and represented in axis-angle, $\boldsymbol{\theta_t}$, (equation 3.4), while removing the offset rotation of the first frame through the quaternion conjugate, $\boldsymbol{(q_0)^c}$. This provides the head orientation as a relative rotation for each system (Figure 4.15). Through the cross-correlation of both signals, $\boldsymbol{t(0)^{Vicon}_{Awinda}}$ is obtained, which allows to synchronize the Awinda suit with the Vicon system.

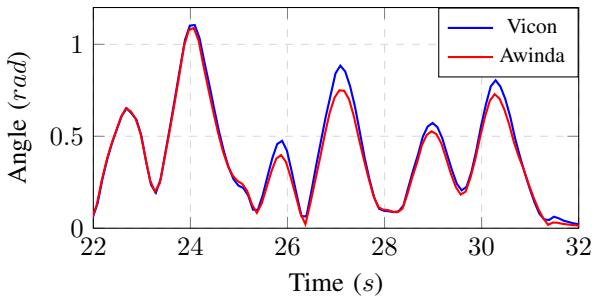


Figure 4.15: Axis-angle representation of both Vicon and Awinda head rotation, giving the delay between systems $\boldsymbol{t(0)^{Vicon}_{Awinda}}$.

4.1.2.3 Toolchain temporal alignment

The definitions in equations 4.5, 4.6 and 4.7 simplify the variable groups from raw ADTF data, $\boldsymbol{n}$, to synchronized, $\boldsymbol{t'}$, for the three ADTF filter samples recorded in Figure 4.4A.

In terms of ToF filter (Algorithm 4.1.2), $\boldsymbol{amp_{t',x,y}}$ represents the synchronized $\boldsymbol{t'}$ amplitude frame considering the raw one, $\boldsymbol{AMP_{n,f}}$, with $\boldsymbol{f}$ being the frame 1D concatenated pixel index; $\boldsymbol{pc_{t',f}}$=$\boldsymbol{\{pcx_{t',f},pcy_{t',f},pcz_{t',f}\}}$ represents the synchronized 3D point cloud from $\boldsymbol{PC_{n,v}}$ with $\boldsymbol{v}$ being the frame 1D concatenated voxel index; $\boldsymbol{depth_{t',x,y}}$ represents the synchronized depth frame from $\boldsymbol{pcz_{t',f}}$.

$$\begin{aligned} S_{s=tof,n} &= \{Amplitude_{n,f}, PointCloud_{n,v}\} \\ &\equiv \\ \therefore S_{s=tof,n} &= \{AMP_{n,f}, PC_{n,v}\} \\ \therefore \delta_{s=tof,t'} &= \{amp_{t',x,y}, pc_{t',f} = \{pcx_{t',f}, pcy_{t',f}, pcz_{t',f}\}, depth_{t',x,y}\} \end{aligned} \tag{4.5}$$

In terms of MVN Awinda Full Body filter (Algorithm 3.1.5), $\boldsymbol{\kappa_{t',j}}$ represents the synchronized joint, $\boldsymbol{j}$, 3D position and $\boldsymbol{\varphi_{t',j}}$ the quaternion rotation.

$$\begin{aligned} S_{s=awinda,n,j} &= \{bodySegmentID_{n,j}, XYZData_{n,j}, QuatData_{n,j}\} \\ &\equiv \\ \therefore S_{s=awinda,n,j} &= \{B_{n,j}, K_{n,j}, Q_{n,j}\} \\ \therefore \delta_{s=awinda,t',j} &= \{\upsilon_{t',j}, \kappa_{t',j}, \varphi_{t',j}\} \end{aligned} \tag{4.6}$$

Finally, for the Vicon Head Tracker filter (Algorithm 4.1.3), $\boldsymbol{\rho_{t'}}$,$\boldsymbol{\varrho_{t'}}$ represent the synchronized position and rotation of the subject's head from the respective raw ones, $\boldsymbol{T_n}$,$\boldsymbol{R_n}$.

$$
\begin{aligned}
S_{s=vicon,n,} &= \{HeadTranslation_n, HeadRotation_n\} \\
&\equiv \\
\therefore S_{s=vicon,n,} &= \{T_n, R_n\} \\
\therefore \delta_{s=vicon,t'} &= \{\rho_{t'}, \varrho_{t'}\}
\end{aligned}
\tag{4.7}
$$

Using the definitions in equations 4.5, 4.6 and 4.7 for each filter output and with each previous temporal calibration, all recorded systems from a real dataset recording are temporally aligned with each other (equation 4.8). Each calibrated temporal mismatch, $\boldsymbol{t(0)^{ToF}_{Vicon}}$, and $\boldsymbol{t(0)^{Vicon}_{Awinda}}$, serves as a time offset to the timestamp of each system's sample: $\boldsymbol{t_C}$ being the ToF, $\boldsymbol{t_O}$ the Vicon, and $\boldsymbol{t_A}$ the MVN Awinda. In Algorithm 4.1.5, it is shown how each individual system is temporaly aligned, considering the initialization process of adding the time offset.

$$
\begin{aligned}
t(0)^{ToF}_{Awinda} &= t(0)^{ToF}_{Vicon} + t(0)^{Vicon}_{Awinda} \\
t_O &= t_O + t(0)^{ToF}_{Vicon} \\
t_A &= t_A + t(0)^{ToF}_{Awinda} \\
t_C &= t_C
\end{aligned}
\tag{4.8}
$$

Algorithm 4.1.5 REAL DATASET SYNCHRONIZATION

initialize:

$Eq\ 4.8$

Get ToF timestamp range T_{range}

$T_{ini} \leftarrow \lambda_t \cap S_{tof,0}$

$T_{end} \leftarrow \lambda_t \cap S_{tof,N}$

$T_{range} \leftarrow T_{end} - T_{ini}$

$T \leftarrow N, S_{s=tof,n}$

Create new synchronization timestamps $\Lambda_{t'}$ and samples $\delta_{s,t'}$ for all systems

$\Lambda_{t'} \leftarrow \lambda_t \cap S_{tof,0 \rightarrow N}, t' = 0, .., T$

$\delta_{s,t'}, t' = 0, .., T$

Interpolate new samples for Awinda $\delta_{s=awinda,t',j}$ and Vicon $\delta_{s=vicon,t'}$

for each t' in $\Lambda_{t'}$ **do**

#MVN Awinda joint position $\kappa_{t',j}$ and rotation $\varphi_{t',j}$

$t_{ini} \leftarrow \lambda_t \cap S_{s=awinda,n,j}, t < t'$

$t_{end} \leftarrow \lambda_t \cap S_{s=awinda,n,j}, t > t'$

$ratio \leftarrow \frac{t'-t_{ini}}{t_{end}-t_{ini}}$

for each j in $\delta_{s=awinda,t',j}$ **do**

$q_{ini} \leftarrow Q_{n,j}, t < t'$

$q_{end} \leftarrow Q_{n,j}, t > t'$

$\varphi_{t',j} \leftarrow SLERP(q_{ini}, q_{end}, ratio)$

$p_{ini} \leftarrow K_{n,j}, t < t'$

$p_{end} \leftarrow K_{n,j}, t > t'$

$\kappa_{t',j} \leftarrow (p_{end} - p_{ini}) \cdot ratio + p_{ini}$

end for

#Vicon Head Tracker position $\rho_{t'}$ and rotation $\varrho_{t'}$

$t_{ini} \leftarrow \lambda_t \cap S_{s=vicon,n,}, t < t'$

$t_{end} \leftarrow \lambda_t \cap S_{s=vicon,n,}, t > t'$

$ratio \leftarrow \frac{t'-t_{ini}}{t_{end}-t_{ini}}$

$q_{ini} \leftarrow R_n, t < t'$

$q_{end} \leftarrow R_n, t > t'$

$\varrho_{t'} \leftarrow SLERP(q_{ini}, q_{end}, ratio)$

$p_{ini} \leftarrow T_n, t < t'$

$p_{end} \leftarrow T_n, t > t'$

$\rho_{t'} \leftarrow (p_{end} - p_{ini}) \cdot ratio + p_{ini}$

#ToF sensor point cloud $pcx_{t',f}, pcy_{t',f}, pcz_{t',f}$

for $v = 1$, v+=3, while $v < 352 \cdot 287 \cdot 3$ **do**

$f \leftarrow f + 1$

$pcx_{t',f} \leftarrow PC_{n,v}$

$pcy_{t',f} \leftarrow PC_{n,v+1}$

$pcz_{t',f} \leftarrow PC_{n,v+2}$

end for

#ToF sensor amplitude $amp_{t',x,y}$ depth $depth_{t',x,y}$

for $x = 1$, x++, while $y < 352$ **do**

for $y = 1$, y++, while $y < 287$ **do**

$f \leftarrow x + (y-1) \cdot 352$

$amp_{t',x,y} \leftarrow AMP_{n,f}$

$depth_{t',x,y} \leftarrow pcz_{t',f}$

end for

end for

end for

end for

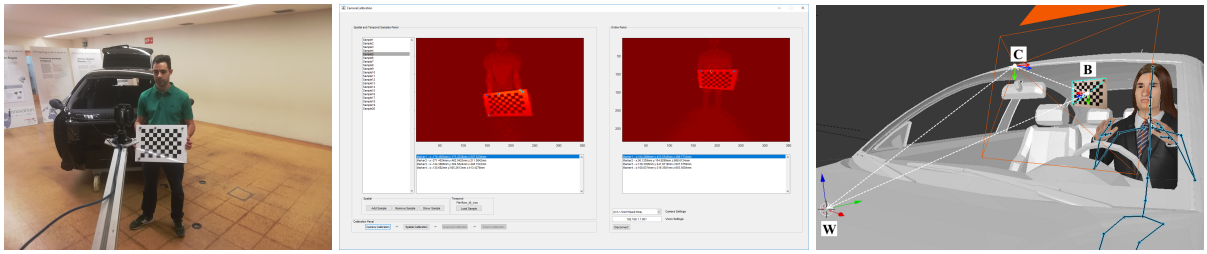
4.1.2.4 Vicon-to-ToF spatial alignment

Vicon Head to Vicon World (T_O^W)

The $\boldsymbol{T_O^W}$ transformation is automatically recorded through the Vicon system, and requires the setup of a head object with a fixed pattern of markers as it was shown in Section 4.1.1.3. $\boldsymbol{T_O^W}$ is the transformation matrix formed with the synchronized position and rotation information from the subject's head $\boldsymbol{\rho_{t'}}$,$\boldsymbol{\varrho_{t'}}$.

Vicon World to ToF Camera (T_W^C)

For the calibration of T_W^C, a checkerboard pattern was used together with four Vicon markers, $B = 1, ..., 4$, placed in its surface (Figure 4.16A). Several samples are taken (Figure 4.16B), and for each, two object plane surfaces were generated for both the ToF, T_B^C, [99] and Vicon, T_B^W, systems (Figure 4.16C).



A B C

Figure 4.16: Vicon world to ToF camera calibration. (A) checkerboard patern with Vicon markers visible to the ToF camera and Vicon system. (B) matlab toolchain for static Vicon and ToF data capture and T_W^C calculation. (C) 3D representation of the coordinate systems when calibrating T_W^C. W: Vicon global coordinate system; C: ToF optical center; B: Checkerboard object plane surface.

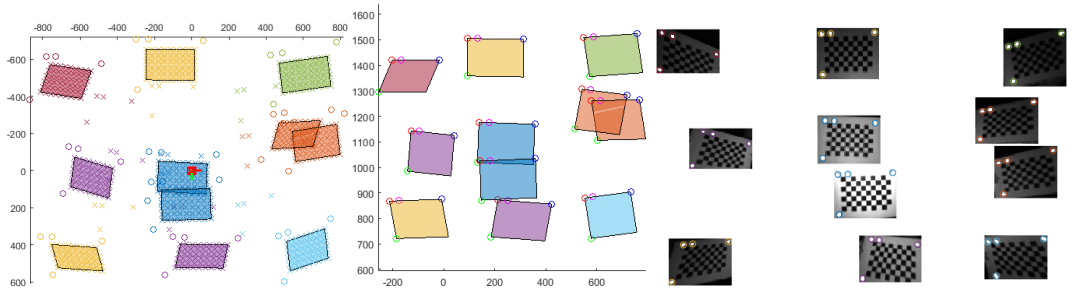
This information allowed to calculate the transformation between systems through a least square minimization problem (equation 4.9), where in order to compute the optimal rotation matrix [100], first both sets of marker, B, 3D coordinates P_B^C and P_B^W must be translated, so that their centroid coincides with the origin of the coordinate system p_B^C and p_B^W. This is done by subtracting from the point coordinates the coordinates of the respective centroid μP_B^C and μP_B^W. Second step consists of calculating a covariance matrix, cvM, followed by the Single Value Decomposition terms V,U, followed by the determinant, dt, of both terms to ensure right-handed coordinate system. Finally, the optimal rotation is calculated, R_W^C, and then added to the translation component in order to calculate the transformation matrix between Vicon world and ToF camera, T_W^C, where I is a 4x4 identity matrix.

$$
\begin{aligned}
p_B^C &= P_B^C - \mu P_B^C \\
p_B^W &= P_B^W - \mu P_B^W \\
cvM &= (p_B^W)^\intercal \cdot p_B^C \\
cvM &= V \cdot S \cdot U^\intercal \\
dt &= det(V \cdot U^\intercal) \\
R_W^C &= U \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix} \cdot V^\intercal \\
T_B^C &= p_B^C \cdot I \\
T_B^W &= p_B^W \cdot I \\
\therefore T_W^C &= T_B^C \cdot R_W^C \cdot T_B^W
\end{aligned}
\tag{4.9}
$$

Algorithm 4.1.6 shows the entire process of $\boldsymbol{T_W^C}$ spatial calibration. First, the static amplitude frames, $\boldsymbol{amp_{t',x,y}}$, obtained in MATLAB give the visible markers position (2D $\boldsymbol{(Pxy_B^C)_{t'}}$ and 3D $\boldsymbol{(P_B^C)_{t'}}$ [Figure 4.17A]) wrt. the ToF camera. Second, the static samples from Vicon give the four markers' 3D position in each timestamp, $\boldsymbol{(P_B^W)_{t'}}$ (Figure 4.17B). Finally, this information from each system is used in equation 4.9 to calculate $\boldsymbol{T_W^C}$ (Figure 4.17C shows $\boldsymbol{(P_B^W)_{t'}}$ being projected into ToF, using $\boldsymbol{T_W^C}$).

Algorithm 4.1.6 T_W^C Calibration

1: # Get P_B^C from amplitude images
2: # Detect ToF markers $(Pxy_B^C)_{t'}$ from amplitude images $amp_{t',x,y}$
3: $(Pxy_B^C)_{t'} \leftarrow Eq.\ 4.2(amp_{t',x,y}), (Pxy_B^C)_{t'} = \{x, y\}$
4: # Convert ToF 2D markers $(Pxy_B^C)_{t'}$ to 3D markers $(P_B^C)_{t'}$
5: $(P_B^C)_{t'} \leftarrow ((Pxy_B^C)_{t'})^\intercal \cdot IM_C, (P_B^C)_{t'} = \{x, y, z\}$(**Figure 4.17A**)
6: # Get P_B^W from Vicon markers
7: $(P_B^W)_{t'} = \{x, y, z\}$(**Figure 4.17B**)
8: # Get T_W^C from ToF 3D markers $(P_B^C)_{t'}$ and Vicon markers $(P_B^W)_{t'}$
9: $T_W^C \leftarrow Eq.\ 4.9(P_B^C \leftarrow (P_B^C)_{t'}, P_B^W \leftarrow (P_B^W)_{t'})$



A B C

Figure 4.17: Vicon world to ToF camera calibration frames: (A) markers 3D position $(P_B^C)_{t'}$ detected in amplitude frames $amp_{t',x,y}$, (B) markers 3D position $(P_B^W)_{t'}$ detected in Vicon system, and (C) markers $(P_B^W)_{t'}$ 2D position in amplitude frames using T_W^C.

4.1.2.5 Awinda-to-Vicon spatial alignment

Awinda Joints to Awinda Head (T_J^A)

The MVN Awinda body pose is tracked inside the MVN global positioning system. This is possible through specific proprietary algorithms that estimate the body gait movement. To minimize sources of error from the MVN Awinda system, one chose to use the relative body pose, removing the global positioning information and its associated errors. To convert the global body pose into a relative one, a joint of reference (root joint) needs to be defined. In this case, the head joint was chosen given its direct relation with the Vicon's head object. To determine the relative position/orientation of each joint wrt. the head, T_J^A, equation 4.10 is used, with R_A, P_A being the Awinda head joint rotation/position wrt. the Awinda world coordinate system, and R_J,P_J the Awinda joint rotation/position (J representing each of the 18 body joints) wrt. the Awinda world coordinate system. In its essence, T_J and T_A are comprised by the recorded rotation quaternions $\varphi_{t',j}$, $\varphi_{t',head}$, and translation $\kappa_{t',j}$, $\kappa_{t',head}$ that were synchronized in Section 4.1.2.3.

$$T_J = \begin{bmatrix} & R_J & & P_J \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

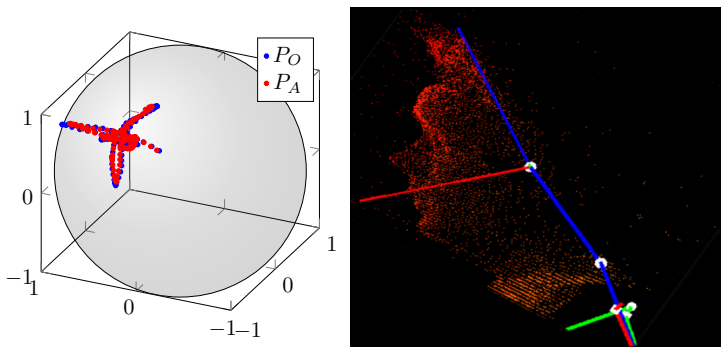
$$\therefore T_J^A = (T_A)^\intercal \cdot T_J$$

Awinda Head to Vicon Head (T_A^O)

Considering the body pose as being relative to the head (root joint), the alignment between the MVN Awinda head and the Vicon head needs to be determined. To determine the $\boldsymbol{T_A^O}$ transformation, each recording procedure is initiated by a small recording of head rotations in each axis (i.e. flexion/extension, lateral flexion and rotation). This information is then used to generate a set of points for the Vicon head object $\boldsymbol{P_O}$, and the Awinda head joint, $\boldsymbol{P_A}$ (equation 4.11).

$$\begin{aligned} P &= (0, 0, 1) \\ P_A &= R_A \cdot P \\ P_O &= R_O^W \cdot P \end{aligned} \tag{4.11}$$

Each set of points $\boldsymbol{P_O}$, $\boldsymbol{P_A}$ corresponds to a vector $\boldsymbol{P}$ being rotated in each timestamp $\boldsymbol{\Lambda_{t'}}$ with the head rotation $\boldsymbol{R_A = \varphi_{t',head}}$, $\boldsymbol{R_O^W = \varrho_{t'}}$ at the specific timestamp. Through the Finite Iterative Closest Point (FICP) algorithm created by Kroon [98], it is possible to find the optimal rotation, $\boldsymbol{R_A^O}$, that best aligns both point clouds that represent the head joint from both systems (Figure 4.18A). With the extra manual translation of the Vicon head position to the real head position $\boldsymbol{P_A^O}$ (C1 vertebrae) as it is shown in Figure 4.18B, the transformation matrix is obtained $\boldsymbol{T_A^O}$.



A B

Figure 4.18: Awinda head to Vicon head spatial alignment, $\boldsymbol{T_A^O}$. (A) shows the rotation $\boldsymbol{R_A^O}$ alignment, and (B) shows the translation $\boldsymbol{P_A^O}$ alignment.

4.1.2.6 Toolchain spatial alignment

With each previous spatial calibration, the human body pose, $\boldsymbol{T_J}$, can be projected into the ToF camera's prespective (equation 4.12).

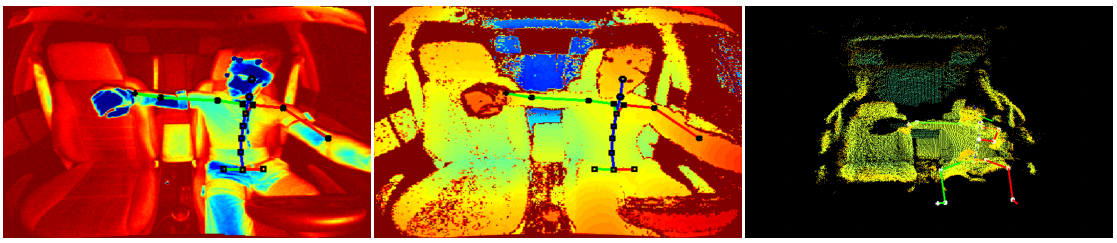
$$\therefore T_J^C = T_W^C \cdot T_O^W \cdot T_A^O \cdot T_J^A \tag{4.12}$$

$\boldsymbol{T_J^C}$ is the combined transformation matrix that spatially aligns each joint from the inertial suit with the camera world, with $\boldsymbol{T_J^A}$ being the transformation that maps each joints' position/orientation wrt. to the head in the inertial suit system, $\boldsymbol{T_A^O}$ the transformation that maps the head joint from the inertial suit system to the Vicon one, $\boldsymbol{T_O^W}$ the position/orientation of the head object within the Vicon's world coordinate system, $\boldsymbol{T_W^C}$ the transformation that maps the Vicon's system to the ToF sensor coordinate system.

4.1.3 Rendering (MATLAB)

After data alignment, the toolchain starts rendering the dataset information (Figure 4.19), including: (A) an amplitude frame; (B) a depth frame; and (C) a 3D point cloud. Moreover, the 2D and 3D body pose of each human model is generated and exported in JavaScript Object Notation (JSON) format. Each exported dataset $\boldsymbol{RD_{t'}}$ is comprised by the group of dataset information (equation 4.13) for each frame $\boldsymbol{t'}$.

$$\begin{aligned} \therefore RD_{t'} = \{&amp_{t',x,y}, depth_{t',x,y}, \\ &pc_{t',f} = \{pcx_{t',f}, pcy_{t',f}, pcz_{t',f}\}, \\ &(Pxy_J^C)_{t'}, (pxyzJC)_{t'}\} \end{aligned} \tag{4.13}$$



A B C

Figure 4.19: Rendered frames: (A) amplitude, (B) depth, (C) point cloud. Images (A, B, C) are represented in color for better visualization.

4.1.3.1 Amplitude frame

The amplitude frame (Figure 4.19A) $\boldsymbol{amp_{t',x,y}, x = 1,...,X, y = 1,...,Y}$, is rendered with the information recorded from the ToF camera. Each pixel information reflects the intensity from the object projected on the pixel in the camera's $\boldsymbol{XY}$ plane.

4.1.3.2 Depth frame

The depth frame (Figure 4.19B) $\boldsymbol{depth_{t',x,y}, x = 1,...,X, y = 1,...,Y}$, where $\boldsymbol{X}$ is the camera's horizontal resolution and $\boldsymbol{Y}$ its vertical resolution, is rendered with the information recorded from the ToF camera,

with each pixel representing the distance from the object to the projected pixel in the camera's $\boldsymbol{XY}$ plane.

4.1.3.3 Point cloud

The 3D point cloud (Figure 4.19C) has the Cartesian coordinates xyz of the voxel that was projected in each pixel, $\boldsymbol{pcx_{t',f}, pcy_{t',f}, pcz_{t',f}}$, $\boldsymbol{f = 1, ..., X \times Y}$.

4.1.3.4 Ground-Truth

The ground-truth information (black squares in Figure 4.19) consists in exporting the pose information for the human using the ground-truth standard in Table 4.1, with respect to the perspective of each frame (equation 4.14). With that in mind, two types of ground-truth are possible: 2D pose for amplitude and depth frames; and 3D pose for the point cloud. Both types of ground-truth consist in the same pose information for the human, that is a structure comprised of all joints' pixel, $\boldsymbol{Pxy_J^C}$ (2D), or voxel, $\boldsymbol{Pxyz_J^C}$ (3D), positions.

$$\begin{aligned} Pxyz_J^C &= P_J^C \\ Pxy_J^C &= Pxyz_J^C \cdot IM_C \end{aligned} \tag{4.14}$$

Table 4.1: Ground-truth joint label correspondance with MVN simplified ISB model from Table 2.1 used in ADTF filter from Algorithm 3.1.5.

Ground-truth joints (Index)	**Simplified ISB (Index)**	**Ground-truth joints (Index)**	**Simplified ISB (Index)**
head (1)	$Head(6)$	rhand (11)	
neck (2)	$Neck(5)$	hip (12)	$L5(0)$
chest (3)	$T8(4)$	lpelvis (13)	$LeftUpperLeg(19)$
lshoulder (4)	$LeftUpperArm(12)$	rpelvis (14)	$RightUpperLeg(15)$
rshoulder (5)	$RightUpperArm(8)$	lknee (15)	$LeftLowerLeg(20)$
lelbow (6)	$LeftForearm(9)$	rknee (16)	$RightLowerLeg(16)$
relbow (7)	$RightForearm(13)$	lankle (17)	$LeftFoot(21)$
lwrist (8)	$LeftHand(14)$	rankle (18)	$RightFoot(17)$
rwirst (9)	$RightHand(10)$	lfoot (19)	$LeftToe(22)$
lhand (10)		rfoot (20)	$RightToe(18)$

4.2 Evaluation results

In this section the toolchain capability to generate real datasets is evaluated. A second evaluation is performed in terms of human body pose estimation application. All MVN Awinda sensors were updated to V4.2.1. Motion capture was performed with MVN Studio 4.97.1. Head tracking was performed with Vicon Nexus 2.7.1 and DataStream SDK V1.2.0. Image capture was performed with libroyale V3.4.0.0.

4.2.1 Toolchain

4.2.1.1 Dataset generation

To evaluate the generation of real datasets, two experiments were established: (1) tightly controlled evaluation for the same sequence of driving actions for both outside and inside the car (TE1 and TE2, respectively), while avoiding any type of collision between sensors/straps and external materials; (2) free movement evaluation for three in-car seated positions (FE1, FE2 and FE3 corresponding to front passenger, driver and back passenger, respectively), while allowing the subject to interact freely with the scenery. For both evaluation procedures, the head quaternion behavior for both systems is extracted (Figures 4.20 and 4.21), associated with a full dataset output from specific timestamps (Figures 4.22 and 4.23).

For the 1^{st} evaluation procedure, the dataset timestamps represent the same actions for inside and outside the car (Figure 4.22). It is possible to see that extreme head rotations increase the body pose error due to the head alignment. It is also possible to observe that magnetic distortions are not significant, where the same actions inside the car can project a body pose similar or better then outside of it.

For the 2^{nd} evaluation procedure, the dataset timestamps represent the best and worst alignments (Figure 4.23). From the analysis, it is possible to pinpoint the main sources of error that contribute to a sub-optimal body pose joint projection:

- Sources of error identified in the inertial suit evaluation (see Chapter 3);
- Absence of projection, due to occlusion of the Vicon's head object;
- High projection error for joints further away from the head joint, due to bad alignment between head objects (indicated with red highlight in Figure 4.21);
- High projection error for the lower body part, due to sensors and straps' movement as a result of collisions with seat, car door or steering wheel.

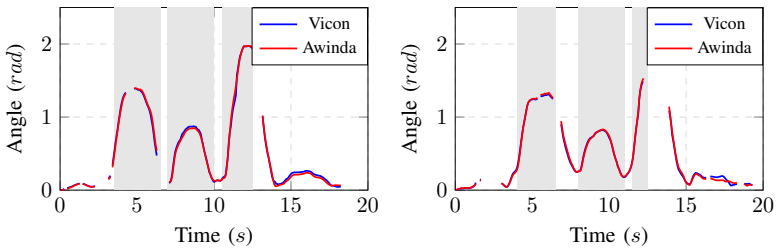


Figure 4.20: Axis-angle representation between Vicon's head object and Awinda's head segment: (A) TE1; (B) TE2. Gray regions highlight the 1^{st}, 2^{nd} and 3^{rd} head maximum rotations for each simulated action.

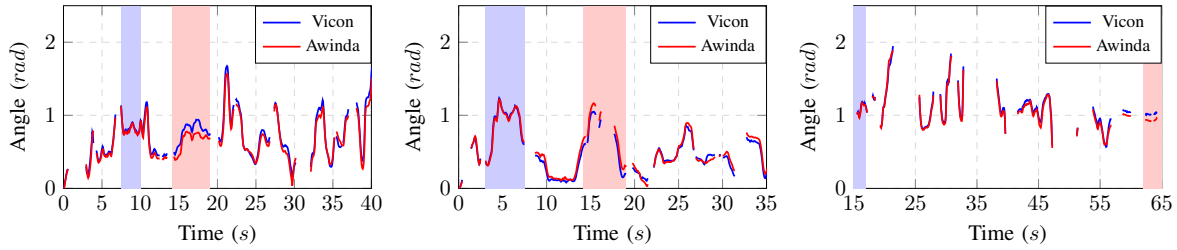


Figure 4.21: Axis-angle representation between Vicon's head object and Awinda's head segment: (A) FE1; (B) FE2; (C) FE3. Blue region highlights the best alignment between both objects, while red region highlights worst alignment.

Figure 4.22: Real dataset frames with associated body pose ground-truth. Representation is done considering the head maximum rotations for each simulated action in each evaluation (Figure 4.20).

Figure 4.23: Real dataset frames with associated body pose ground-truth. Representation is done considering the best and worst head alignment in each evaluation (Figure 4.21).

4.2.2 Application to Pose Estimation Problems

To understand the validity of the data being generated with the toolchain, as well as its ability to increase ML algorithmic accuracy, three distinct experimental scenarios were defined: 2D pose estimation from depth images; 2D pose estimation from point cloud; and 3D pose estimation from 2D pose.

4.2.2.1 Evaluation Data

Since the aim is to evaluate the potential advantage of using an in-car focused dataset over a generic one, samples from the system and from publicly available datasets are required. In this sense, we used two publicly available datasets, the first being Towards Viewpoint Invariant 3D Human Pose Estimation (ITOP) [62], containing 17991 real images and corresponding ground-truth from a single subject, $\boldsymbol{S_1}$; the second was the NTU RGB+D [101], where the first subject, $\boldsymbol{Z_1}$ and all of its planes, $\boldsymbol{P}$, cameras, $\boldsymbol{C}$, rotations, $\boldsymbol{R}$ and 49 actions, $\boldsymbol{A_{1:49}}$ were used, resulting in 94321 real images and corresponding ground-truth. Our toolchain generated dataset consists in five recorded subjects, $\boldsymbol{H_{1:5}}$, two actions, $\boldsymbol{B_{1:2}}$ each, totaling 8754 samples (i.e. recording compliant with General Data Protection Regulation [GPRD] [Appendix A]).

All datasets are identical in terms of sample data types for depth frame and 3D/2D body pose, giving us the opportunity to evaluate the first and third experimental scenarios comparitively to each other. The second experimental scenario permits the assessment of the proposed dataset to estimation problems based on point clouds. The available samples were divided into 3 groups: (1) a training set, with all public datasets plus 6322 toolchain samples (from subjects $\boldsymbol{S_{1:4}}$); (2) a validation set with 702 toolchain samples (from subjects $\boldsymbol{S_{1:4}}$); and (3) a test set with 1730 toolchain samples (from subject $\boldsymbol{S_5}$ performing distinct actions). To assess the influence of mixing public datasets with the toolchain generated ones, seven sub-evaluations were established, $\boldsymbol{M_{1:7}}$, for the first and third experiment (Table 4.2). For the second experiment $\boldsymbol{M_1}$ sub-evaluation was used.

The proposed toolchain generated dataset, plus the tools needed to reproduce all experiments, were made publically available [102].

Table 4.2: Evaluations related with toolchain and public data quantities. Each ***M#*** represents a sub-evaluation for the assessment of the influence of mixing public datasets with the toolchain generated one.

Evaluation	MoLa R8.7k InCar	ITOP	NTU RGB+D
$M1^1$	6322	0	0
$M2^1$	0	0	94321
$M3^2$	6322	0	94321
$M4^1$	6322	0	94321
$M5^1$	0	17991	0
$M6^2$	6322	17991	0
$M7^1$	6322	17991	0

[1] 200 epochs for the full dataset. [2] 180 epochs for the public dataset plus 20 for fine-tuning with the toolchain dataset

2D Pose Estimation from Depth Images (RE1)

To evaluate the toolchain generated depth frames and corresponding 2D ground-truth, the Part Affinity Fields (PAF) [25] method was used. From it, a custom Convolutional Neural Network (CNN) was implemented consisting only on the first stage of the original PAF CNN. In each sub-evaluation, ***M#***, the method used the depth frame as input features and the 2D body pose as output labels (Figure 4.24). For all samples, the depth frame was normalized into a grayscale frame ($[0; 1.8]\ m \equiv [0; 255]$), while each 2D joint position was converted into a 2D heatmap. For metric evaluation, the joint position is estimated through non-maximum suppression applied to the inferred heatmap. In this experiment, the PCKh measure (in pixels, using a matching threshold given by 50% of the head segment length) and the Area Under Curve (AUC) were used as metrics [103]. Table 4.3 summarizes the average results for the full body, with the results for individual joints being presented in Table 4.4. Figure 4.25A presents the PCKh@0.5 values for the full body for each sub-evaluation.

2D Pose Estimation from Point cloud (RE2)

To evaluate the toolchain generated point cloud and corresponding 2D ground-truth, the PAF [25] method was used. In this experiment, the point cloud was used as input features (Figure 4.24). To this end, each point cloud was normalized (pcx and pcy with $[-1.5; 1.5]\ m \equiv [0; 255]$, and pcz with $[0; 1.8]\ m \equiv [0; 255]$) and converted into a 3-channel matrix. As for RE1, the network's output was the 2D heatmaps generated from each joint's position, with the inferred joint position being computed by non-maximum suppression. The

same metrics from RE1 were employed. Results are shown in Tables 4.3, 4.4, and Figure 4.25B.

3D Pose Estimation from 2D Pose (RE3)

To evaluate the toolchain generated 3D ground-truth, a 3D pose estimation method [104] was used. The method uses a 2D body pose as input features (provided as joint pixel coordinates) and the 3D body pose as output (Figure 4.24). Once again, similar metrics were employed, but in this case PCKh matching threshold was normalized to a fixed head size of $200\ mm$. Results are shown in Tables 4.3, 4.4, and Figure 4.25C.

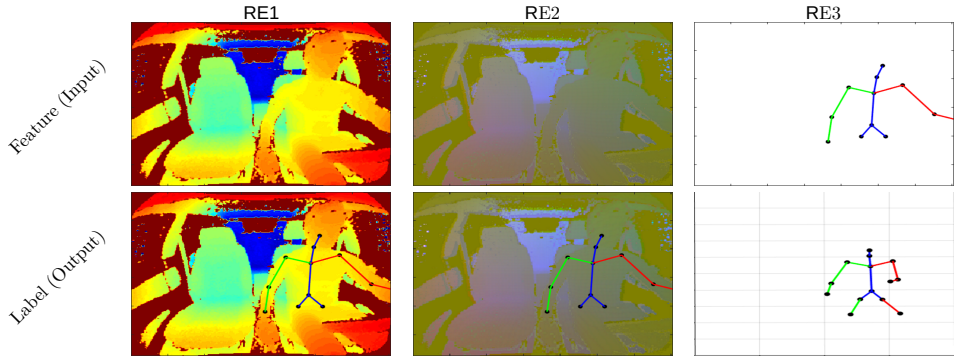


Figure 4.24: Visual representation of input features and output used for each experimental scenario $RE\#$: (RE1) 2D pose estimation from depth images using normalized depth frame as input and 2D body pose as output; (RE2) 2D pose estimation from point cloud using normalized point cloud as input and 2D body pose as output; and (RE3) 3D pose estimation from 2D pose using 2D body pose as input and 3D body pose as output.

Table 4.3: PCKh measure and AUC values averaged over all 14 joints, for the 3 experimental scenarios and all 7 sub-evaluations. Each ***M#*** represents a sub-evaluation for the assessment of the influence of mixing public datasets with the toolchain generated one. Each ***RE#*** represents different pose estimation scenarios.

		M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**
RE1	PCKh[1]	95.97	0.01	69.54	94.88	12.64	94.57	95.48
	AUC	56.14	0.01	33.36	59.39	5.26	58.04	55.33
RE2	PCKh[1]	96.47						
	AUC	64.43						
RE3	PCKh[2]	95.53	0.00	95.12	97.25	0.00	95.80	97.73
	AUC	59.87	0.00	56.81	57.68	0.00	62.07	65.63

[1] RE1 and RE2 does matching threshold to $26\ pixels$. [2] RE3 does matching threshold to $200\ mm$.

Table 4.4: Evaluation results (PCKh@0.5) per joint group in the three experimental scenarios. Each ***M#*** represents a sub-evaluation for the assessment of the influence of mixing public datasets with the toolchain generated one. Each ***RE#*** represents different pose estimation scenarios.

		M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**
RE1[6]	Head[1]	99.88	0.00	60.00	100.00	5.00	99.48	94.68
	Shoulder[2]	99.87	0.02	62.89	100.00	0.00	99.87	99.85
	Elbow[3]	97.23	0.00	79.71	99.31	40.87	95.23	97.14
	Wrist[4]	62.99	0.00	19.80	49.14	25.93	44.35	64.34
	Hip[5]	100.00	0.00	93.99	100.00	0.00	99.90	99.75
RE2[6]	Head[1]	99.94						
	Shoulder[2]	100.00						
	Elbow[3]	98.06						
	Wrist[4]	59.38						
	Hip[5]	100.00						
RE3[7]	Head[1]	100.00	0.00	99.64	100.00	0.00	100.00	100.00
	Shoulder[2]	100.00	0.00	100.00	100.00	0.00	99.94	100.00
	Elbow[3]	90.54	0.00	86.39	92.76	0.00	89.06	95.94
	Wrist[4]	82.66	0.00	84.70	90.75	0.00	85.85	90.42
	Hip[5]	100.00	0.00	100.00	100.00	0.00	100.00	100.00

[1] Head uses head and neck joints. [2] Shoulder uses rshoulder, lshoulder and chest joints.

[3] Elbow uses relbow and lelbow joints. [4] Wrist uses rwrist and lwrist joints. [5] Hip uses rhip, lhip and pelvis joints.

[6] RE1 and RE2 does matching threshold to $26\ pixels$. [7] RE3 does matching threshold to $200\ mm$

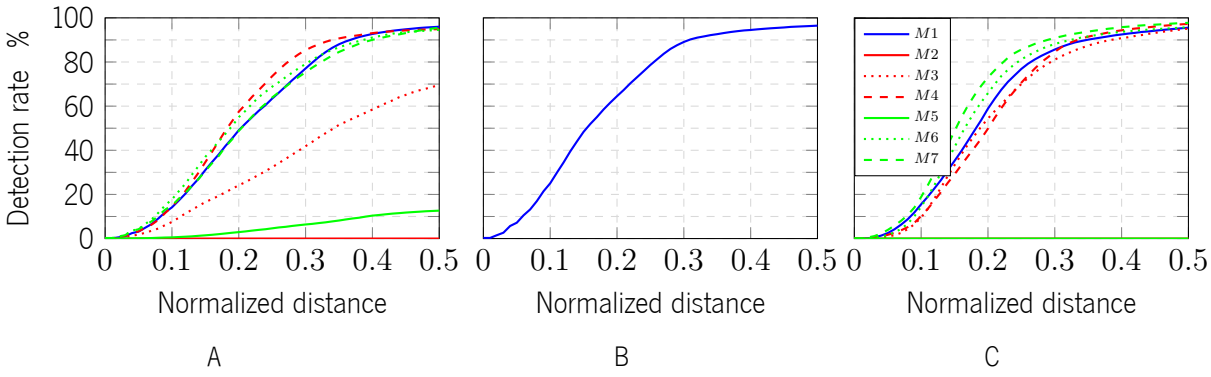


Figure 4.25: PCKh total for all sub-evaluations, $M\#$, and the three first experimental scenarios, $RE\#$: (A) 2D pose estimation from depth images (RE1); (B) 2D pose estimation from point cloud (RE2); and (C) 3D pose estimation from 2D pose (RE3). Color gradient represents different combinations of datasets. Continuous lines represent one dataset trained for 200 epochs, dotted lines represent two datasets trained in sequence (1^{st} $\rightarrow$180 epochs, 2^{nd} $\rightarrow$20 epochs), and dashed lines represent two mixed datasets trained for 200 epochs.

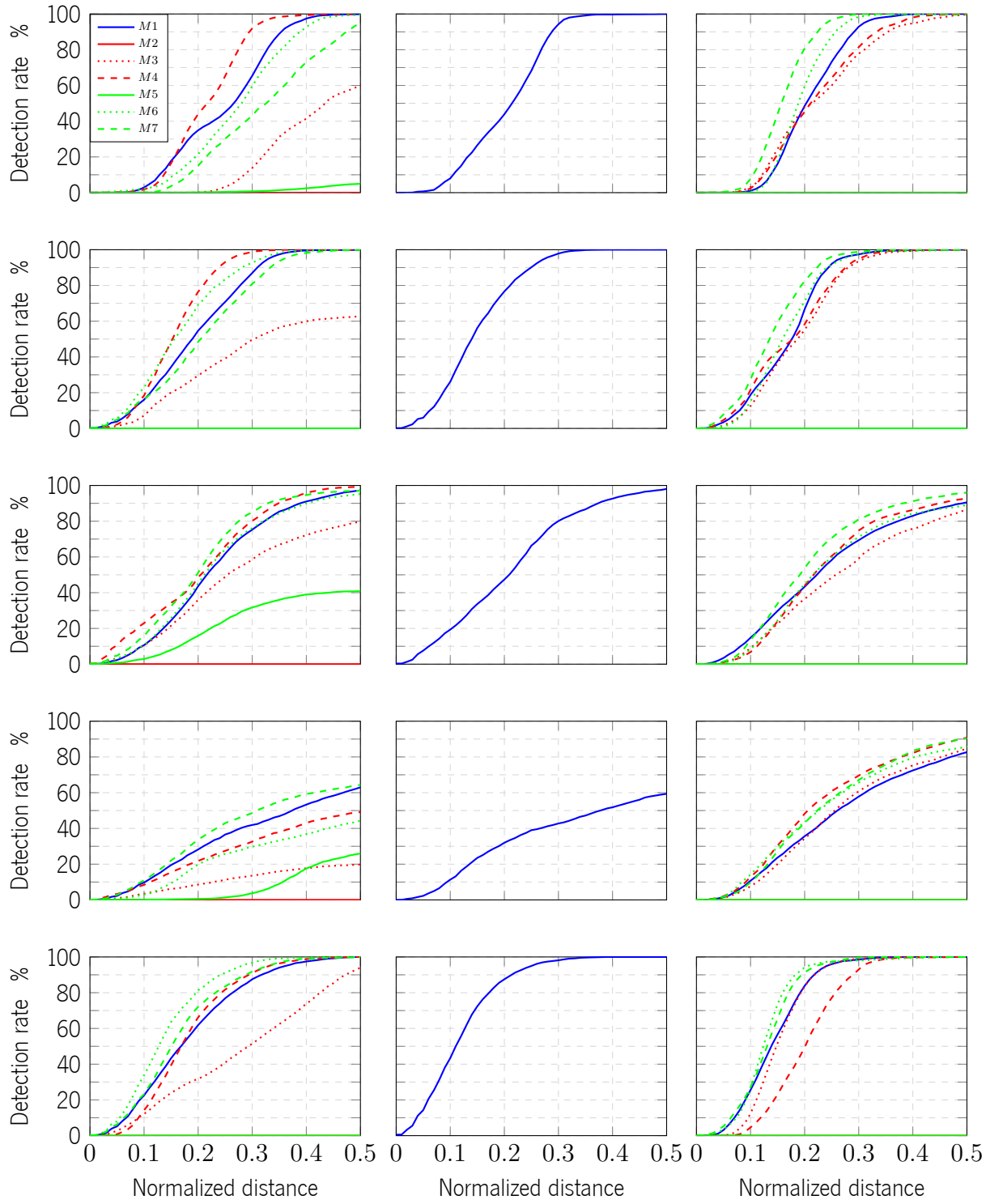


Figure 4.26: PCKh for all sub-evaluations, $M\#$, and the three first experimental scenarios, $RE\#$: (1^{st} column) 2D pose estimation from depth images (RE1); (2^{nd} column) 2D pose estimation from point cloud (RE2); (3^{rd} column) 3D pose estimation from 2D pose (RE3); (1^{st} row) Head; (2^{nd} row) Shoulder; (3^{rd} row) Elbow; (4^{th} row) Wrist; and (5^{th} row) Hip. Color gradient represents different combinations of datasets. Continuous lines represent one dataset trained for 200 epochs, dotted lines represent two datasets trained in sequence (1^{st} $\rightarrow$180 epochs, 2^{nd} $\rightarrow$20 epochs), and dashed lines represent two mixed datasets trained for 200 epochs.

4.3 Discussion

In terms of the toolchain output for human body pose detection, it falls behind other methods [51, 49] when used in an open space. The big novelty and advantage of the proposed toolchain is for the in-car scenario. Here, the toolchain improves considerably on others that share the same in-car focus [67]. As mentioned before, two state-of-the-art motion capture systems (optical and inertial) were fused. By doing it, their stand-alone limitations were suppressed (marker occlusion, global positioning drift) and their added benefits increased by creating a motion capture system for highly occluded scenarios. It was possible to record and project a human motion capture system into an image sensor in an heavily occluded scenario. The projection is possible through specific calibration procedures that allow for a temporal and spatial alignment of all recorded systems. Due to this complex pipeline, several sources of error exist, with the major ones being associated to the inertial motion capture suit (Figure 3.15) and the Awinda to Vicon head spatial alignment (Figure 4.21). Despite this calibration sensitivity, it was possible to record in-car datasets with proper human body pose motion capture (Figure 4.23). The toolchain shows robustness to magnetic distortion scenarios, namely inside the vehicle, where it was possible to observe a performance similar to movements performed outside the vehicle (Figure 4.22). The toolchain can also be applied in other scenarios where ambient occlusion is a limitation factor for motion capture.

In terms of data validation, Figures 4.25, 4.26 and Table 4.3 demonstrate the interest in using toolchain generated data in ML training for the in-car scenario, showing PCKh improvements in all experiments when adding toolchain generated data. $\boldsymbol{RE1{:}M1}$ and $\boldsymbol{RE2{:}M1}$ proved that training with specific use-case datasets can achieve best accuracy (in-car scenario). On the other end, mixed dataset combinations (i.e. ITOP or NTU RGB+D plus MoLa 8.7k InCar) also performed better then single generic dataset training (e.g. $\boldsymbol{M2 < M4}$ and $\boldsymbol{M5 < M7}$) demonstrating again the interest of using specific use-case datasets for training. Evaluations that rely on fine-tuning also presented the same results (e.g. $\boldsymbol{M2 < M3}$ and $\boldsymbol{M5 < M6}$), meaning that pre-trained models can be fine-tuned with the toolchain's dataset to achieve good results. Interestingly, besides demonstrating the interest of toolchain generated data for increased algorithmic accuracy, the present results also seem to suggest that the use of a 3D point cloud as input may lead to a better pose inference when compared to networks using depth images (see for example the higher AUC values for $\boldsymbol{RE2{:}M1}$ compared with $\boldsymbol{RE1{:}M1}$). In the end, for the in-car 2D body pose estimation problem, a higher PCKh@.5 total score in $\boldsymbol{RE2{:}M1}$ (i.e. MoLa 8.7k InCar point cloud based training) of 96.47% and corresponding AUC of 64.43% was achieved. For the in-car 3D body pose estimation problem, the same conclusions can be made with regard to using specific use-case datasets vs. generic ones. However, better

results were also achieved when mixing datasets instead of training with the toolchain generated dataset alone (i.e. $\boldsymbol{RE3{:}M1 < M6 < M7}$). In this case, considering both metrics (PCKh@0.5 total and AUC), it achieved 97.73% and 65.63% respectively for $\boldsymbol{RE3{:}M7}$. Finally, Table 4.3 summarizes the results for individual joints, being possible to conclude that joints frequently present at the image's limits (wrists and elbows) are the most problematic. This may be related with the lower number of training samples with these joints visible (as they are more frequently outside of the camera's field-of-view or in the camera's deadzone). Hip joints also show similar problematic results, but in this case is related to two types of errors (i.e. Awinda-to-Vicon spatial alignment [Section 4.1.2.5], and forward kinematics error propagation from head joint [Chapter 3, Section 3.2.4]), creating less stable ground-truth data for these specific joints.

4.4 Conclusions

In this chapter, a novel toolchain for the generation of in-car human body pose datasets is presented (Appendix B). The toolchain demonstrated to be able to generate datasets through a specific setup consisting in an inertial suit, a global positioning system and a ToF camera, coupled with a set of calibration procedures. The motion capture system was thoroughly evaluated and the sources of error were presented. A toolchain generated dataset is also made publicly available.

In terms of future work, and regarding the calibration procedure, an extra step could be added for the correction of the initial suit calibration, as previously presented in [55]. This would allow a lower sistematic error for the projected body pose. Notwithstanding, this step would have to be non-intrusive for the recording procedure. In terms of dataset quality, there are currently several limitations, mostly coming from the inertial suit (namely related with sensor fixation and soft tissue movement). This could be solved with future inertial suits or better initial calibration procedures from the supplier.

Chapter 5

Synthetic Dataset Toolchain

As shown in Chapter 1, given the aimed development pipeline, this chapter is parallel to Chapter 4 and is a requirement for chapters 6 and 7. Preliminary versions of the work presented in this chapter were presented in paper [81].

In the first section of this chapter, the synthetic dataset toolchain (previously mentioned in Section 1.5.3) overview is shown. The three conceptual modules are then presented: the first module shows the human model creation and its customization aspects (Section 5.1.1); the second module shows how the entire 3D virtual environment is created, generated and validated through specific collision verifications (Section 5.1.2); and the third module shows the entire dataset rendering output, focused on image sensor frames and human body pose ground-truth (Section 5.1.3).

In the second section of this chapter, the synthetic dataset toolchain implementation is shown in-depth. The automation prespective is presented, with context to the software tools used. The mathematical algoritmic concept is shown for the complete toolchain automation, as well as for sub-algorithms, such as collision detection, frame rendering, noise modeling, etc. Algorithmic implementation focuses in mimicking the real dataset toolchain image sensor characteristics (Section 2.1.3.3). Sub-algorithm algorithm complexity is also shown for the more complex ones.

In the third section of this chapter, the entire system's evaluation and potential interest is shown. Two evaluation procedures were defined, one focused on the algorithmic performance of the toolchain, and a second focused on the synthetic data application for human body pose estimation. The first evaluation procedure (Section 5.3.1) evaluates each sub-algorithm complexity, in order to identify maximum toolchain performance in regard to user configurations. The second evaluation procedure (Section 5.3.2) shows how synthetic datasets generated by the toolchain can improve the accuracy of a Machine Learning (ML) based method for human body pose detection.

In the forth and fifth sections of this chapter, discussion, conclusions and future work are shown. Results from evaluation procedures show that the toolchain improves on others, while being the only one focused in the in-car scenario. Toolchain generated data demonstrated to be valuable for ML based methods, increasing their detection accuracy, specifically with the 3D point cloud. Future improvements are discussed, specifically for the most complex sub-algorithms.

Contents

5.1 Toolchain Overview

In this section, the pipeline of the proposed toolchain is described. The proposed pipeline can be divided into three conceptual modules, as illustrated in Figure 5.1. The first module corresponds to the human model creation (Section 5.1.1). In this module, different human models are created through the MakeHuman [105] engine, with the associated skeleton and skin texture. The second module concerns the scene engine (Section 5.1.2). Using as input the human models created in the first stage, this module is responsible for the initialization of the scene considering all objects in it (humans, car and camera), followed by the body pose generation for each human model and associated validation with respect to collisions. Finally, the third module corresponds to the rendering phase (Section 5.1.3), where specific camera frames and human body poses are rendered taking into account the camera perspective and customization. This last module also improves the output data (Time-of-Flight (ToF) images and point cloud) through image processing procedures (namely using the Neural Style Transfer (NST) technique [106]). Both modules (sections 5.1.2 and 5.1.3) are based on Blender and Python programming.

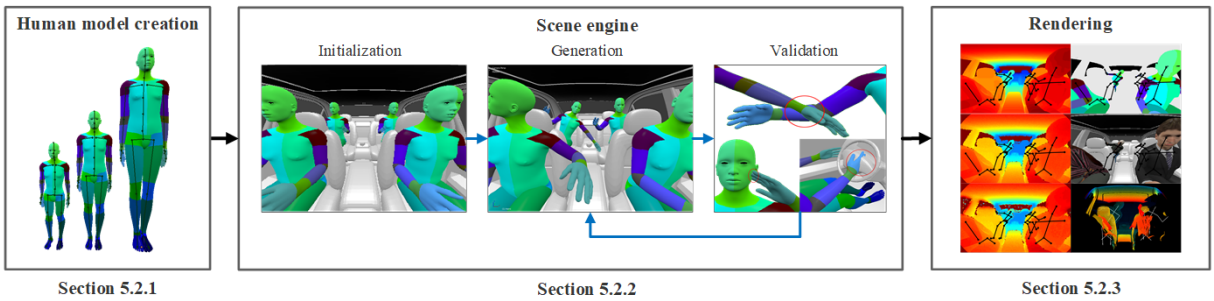


Figure 5.1: Overview of the toolchain pipeline.

5.1.1 Human model creation

The first step of the proposed toolchain is the creation of different human models using the MakeHuman [105] engine. Within this human model creation module, a full customization of the model shape is allowed, through the definition of several parameters, namely body size, body mass, body asymmetry, among others. In fact, the creation of human models with different characteristics, in terms of their shape, is an important task in the toolchain. The higher the variability of the different human models, the higher the quality of the dataset, allowing to give the expected generality for the body pose detection algorithm. This step is fully automated in the toolchain, as the models are randomly created in the initialization step and customized through specific parameters accessible to the user.

Each created human model (equation 5.1), η_h, is comprised of a full body skin mesh, η_h^{sm}, a full

body skin texture, $\boldsymbol{\eta^{st}_{h}}$, and a body skeleton with 6 Degrees-of-Freedom (DoF) per joint, $\boldsymbol{\eta^{bp}_{h,j,a}}$, with joint position/orientation, $\boldsymbol{\eta^{p}_{h,j}}$, $\boldsymbol{\eta^{o}_{h,j,a}}$, and Range of Motion (RoM), $\boldsymbol{\eta^{rm}_{h,j,a}}$, $\boldsymbol{h = 1, ..., H}$, $\boldsymbol{j = 1, ..., Jf}$, $\boldsymbol{a = 1, ..., Af}$, where $\boldsymbol{H}$ represents the number of human models, $\boldsymbol{Jf}$ the number of joints that can have freedom of movement in the model, and $\boldsymbol{Af}$ the axis of freedom on the joint. Finally, a human motion profile is defined within the toolchain, for each human a user input, $\boldsymbol{\Gamma^{H}_{h}}$, is defined (Figure 5.3B) with a Gaussian, $\boldsymbol{\Gamma^{g}_{h,j,a}}$, or incremental, $\boldsymbol{\Gamma^{inc}_{h,j,a}}$, motion profile, an initial body pose, $\boldsymbol{\Gamma^{ip}_{h,j,a}}$, associated to each of its joints, $\boldsymbol{j}$, axes, $\boldsymbol{a}$, and an RGB and label skin texture, $\boldsymbol{\Gamma^{rgb}_{h}}$, $\boldsymbol{\Gamma^{lb}_{h}}$.

$$\begin{aligned} \Gamma^{H}_{h} &= \{\Gamma^{g}_{h,j,a} = \{\Gamma^{\sigma}_{h,j,a}, \Gamma^{\mu}_{h,j,a}\}\Gamma^{inc}_{h,j,a}, \Gamma^{ip}_{h,j,a}, \Gamma^{rgb}_{h}, \Gamma^{lb}_{h}\} \\ \eta_{h} &= \{\eta^{sm}_{h}, \eta^{st}_{h}, \eta^{bp}_{h,j,a} = \{\eta^{p}_{h,j}, \eta^{o}_{h,j,a}\}, \eta^{rm}_{h,j,a}, \eta^{sk}_{h}\} \end{aligned} \tag{5.1}$$

Even though MakeHuman presents different formats for the body skeleton, the Carnegie Mellon University (CMU) format was employed given its relationship with the Vicon kinematic fit model (Chapter 2) and the inherent resemblance with the MVN Awinda Internal Society of Biomechanics (ISB) model (Chapter 2), making it easier to establish a ground-truth format similar to the real dataset (Table 4.1). Each human label skin texture is fully segmented according to the different body parts, where this texture is transversal to all body models, being deformed to fit the different body shapes. The segmented skin labels (i.e. body joints and body segments) has specific RGB color codes, as illustrated in Figure 5.2 and Table 5.1. This skin does not exist when the human model is created (i.e. each human model comes with a generic real skin texture representation), thus its color code and painting needed to be defined/created. Body part segmentation is not used for the final algorithmic development in this thesis, however it is considered a relevant information for current [2, 50] and future methods.

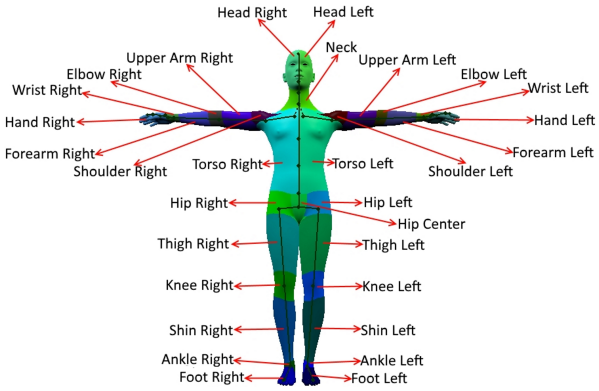


Figure 5.2: RGB body part segmentation of the human model.

Table 5.1: RGB code for body part segmentation of human model.

Label	**RGB**	**Label**	**RGB**	**Label**	**RGB**	**Label**	**RGB**
Foot Left	0x000055	Thigh Left	0x00AA55	Shoulder Left	0x550000	Wrist Left	0x55AA00
Foot Right	0x0000AA	Thigh Right	0x00AAAA	Shoulder Right	0x550055	Wrist Right	0x55AA55
Ankle Left	0x0000FA	Hip Left	0x00AAFA	Upper Arm Left	0x5500AA	Hand Left	0x55AAAA
Ankle Right	0x005500	Hip Right	0x00FA00	Upper Arm Right	0x5500FA	Hand Right	0x55AAFA
Shin Left	0x005555	Hip Center	0x00FA55	Elbow Left	0x555500	Neck	0x55FA00
Shin Right	0x0055AA	Torso Left	0x00FAAA	Elbow Right	0x555555	Head Left	0x55AA55
Knee Left	0x0055FA	Torso Right	0x00FAFA	Forearm Left	0x5555AA	Head Right	0x55AAAA
Knee Right	0x00AA00			Forearm Right	0x5555FA		

5.1.2 Scene engine

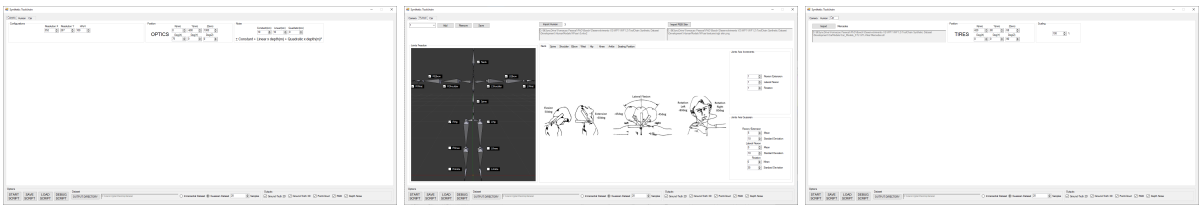
The scene engine module is responsible for the creation of the scene to be rendered. This block is subdivided into three different parts, namely initialization, generation, and validation. These three stages will be explained in detail in the next sub-sections. Moreover, this engine requires user input to create the scenes (equations 5.1, 5.2 and 5.3).

A camera model is chosen (equation 5.2), $\boldsymbol{\iota}$, with given user-defined camera parameters, $\boldsymbol{\Gamma^T}$, (Figure 5.3A), including camera resolution, $\boldsymbol{\Gamma^X}$, $\boldsymbol{\Gamma^Y}$, horizontal Field-of-View (FoV), $\boldsymbol{\Gamma^{HFoV}}$, position, $\boldsymbol{\Gamma^{tp}}$, orientation, $\boldsymbol{\Gamma^{to}}$ and axial noise model, $\boldsymbol{\Gamma^n = \{\Gamma^a, \Gamma^b, \Gamma^c\}}$.

$$\begin{aligned} \Gamma^T &= \{\Gamma^X, \Gamma^Y, \Gamma^{HFoV}, \Gamma^{tp}, \Gamma^{to}, \Gamma^n = \{\Gamma^a, \Gamma^b, \Gamma^c\}\} \\ \iota &= \{\iota^X, \iota^Y, \iota^{HFoV}, \iota^p, \iota^o, \iota^n\} \end{aligned} \tag{5.2}$$

A car model (equation 5.3), $\boldsymbol{\zeta}$, is also given as user input, $\boldsymbol{\Gamma^C}$, to this block (Figure 5.3C), which is imported to create the realistic 3D car model mesh, $\boldsymbol{\Gamma^{cm}}$, with 6DoF position/orientation, $\boldsymbol{\Gamma^{cp}}$,$\boldsymbol{\Gamma^{co}}$, to their equivalent set or variables from the car model, $\boldsymbol{\zeta}$.

$$\begin{aligned} \Gamma^C &= \{\Gamma^{cp}, \Gamma^{co}, \Gamma^{cm}\} \\ \zeta &= \{\zeta^p, \zeta^o, \zeta^m\} \end{aligned} \tag{5.3}$$



A B C

Figure 5.3: C# HMI user input, $\mathbf{\Gamma}$, for scene initialization: (A) camera, $\mathbf{\Gamma^T}$, (B) humans, $\mathbf{\Gamma_h^H}$, and (C) car, $\mathbf{\Gamma^C}$.

5.1.2.1 Initialization

The first step of the initialization stage is to create and position the humans, the camera, and the car model in the synthetic scenario. Regarding the human models, there is an extra step. Every model comes with an unrealistic motion profile, in which each joint is comprised of 3 axis with $\pm$ 180° of range of motion. Considering the real anthropometric constraints of the human body [107], the motion profile is modified. In this sense, each joint axis is restricted to its corresponding human body joint axis (Table 5.2) and RoM, $\boldsymbol{\eta_{h,j,a}^{rm}}$.

Table 5.2: Axis correspondence between MakeHuman model and real human.

Joint	+X	+Y	+Z
C7 Vertebrae	Flexion	-Rotation	-Lateral Flexion
L5 Vertebrae	Flexion	DISABLED	Lateral Flexion
Left Shoulder	Flexion	Rotation	Abduction
Right Shoulder	Flexion	-Rotation	Adduction
Left Elbow	Flexion	Pronation	DISABLED
Right Elbow	Flexion	Supination	DISABLED
Left Wrist	Radial	DISABLED	Extension
Right Wrist	Radial	DISABLED	Flexion
Left Hip	Extension	DISABLED	Adduction
Right Hip	Flexion	DISABLED	Abduction
Left Knee	Flexion	DISABLED	DISABLED
Right Knee	Flexion	DISABLED	DISABLED
Left Ankle	Flexion	Eversion	DISABLED
Right Ankle	Flexion	Inversion	DISABLED

5.1.2.2 Generation

After the initialization stage, a new scene is generated, where the human models perform random movements, where they can have Gaussian or incremental profiles.

If the Gaussian profile is selected, each human has a user-customized motion profile that follows a Gaussian distribution for each joint axis, $\boldsymbol{\Gamma^{g}_{h,j,a}}$. The user is able to customize the distributions' mean value, $\boldsymbol{\Gamma^{\mu}_{h,j,a}}$, and standard deviation, $\boldsymbol{\Gamma^{\sigma}_{h,j,a}}$. For every new frame generated by the toolchain, a new value is sampled from these distributions and applied to each joint axis, $\boldsymbol{\eta^{o}_{h,j,a}}$, in each human model inside the scene.

If in turn the incremental profile is selected, each human comes with a user customized motion profile that follows a truth table of possible poses, considering the selected incremental angle for each joint axis, $\boldsymbol{\Gamma^{inc}_{h,j,a}}$. In the hypothetical situation that the human body would have the same axis of freedom per joint, the same axis range of motion, $\boldsymbol{\eta^{rm}_{h,j,a}}$, and the same incremental angle for all joint axis, the number of possible poses, $\boldsymbol{np_h}$, for each human, $\boldsymbol{h}$, is given by equation 5.4. This method allows to "brute force" the generation of human body poses, by increasing or decreasing the incremental angle per joint axis across the entire set of possible values.

$$\begin{aligned} step_h &= \frac{\eta_h^{rm}}{\Gamma_h^{inc}}, step_h \in \mathbb{N} \\ \therefore np_h &= step_h^{Af \times Jf} \end{aligned} \tag{5.4}$$

5.1.2.3 Validation

After the scene has been generated, it must be validated. The validation process consists in searching for scene collisions. There are three identified collision types that need to be considered to generate realistic datasets. These collision types are illustrated in Figure 5.4. The toolchain automatically checks for all collisions after generating all poses, in order to guarantee the realism of the scene and dataset. If there are no collisions, then the dataset is rendered, otherwise a new scene is generated.

Body to body collisions refer to inner body intersections, and are evaluated for all humans (Figure 5.4A). This detection inspects the body 3D skeleton mesh, $\boldsymbol{\eta^{sk}_{h}}$, (Figure 5.4D) against the human model skin mesh, $\boldsymbol{\eta^{sm}_{h}}$. For this detection, the toolchain automatically creates a 3D skeleton for each human model, $\boldsymbol{\eta^{sk}_{h}}$, using the human model skeleton, $\boldsymbol{\eta^{bp}_{h,j,a}}$, dimensions (joints' positions and segments' lengths).

Human to human collisions refer to human intersections, and are evaluated between all humans in the scene (Figure 5.4B). This detection inspects the human models skin mesh, $\boldsymbol{\eta^{sm}_{h}}$, against each other.

Human to car collisions refer to human to car intersections, and are evaluated for all humans (Figure 5.4C). This detection inspects the human models skin mesh, $\boldsymbol{\eta^{sm}_{h}}$, against the car model mesh, $\boldsymbol{\zeta^{m}}$.

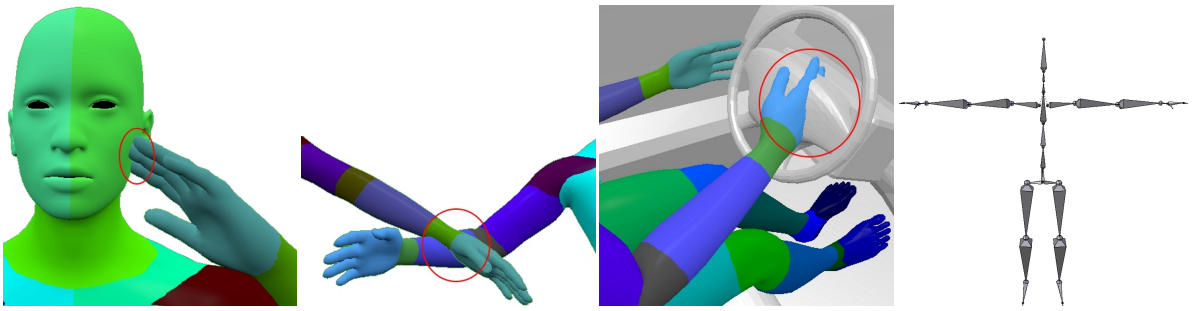


Figure 5.4: Collision detection types: (A) body to body, (B) human to human, (C) human to car. (D) 3D skeleton, η_h^{sk}, created from $\eta_{h,j,a}^{bp}$ segments' lengths and joints' positions to assist in body to body collision detection.

5.1.3 Rendering

After scene validation, the toolchain starts rendering the dataset information through Blender proprietary rendering pipeline (Figure 5.5), and post-processing algorithms (Figure 5.6), including: (1) depth frames (clean, noise and NST); (2) a 3D point cloud; (3) an RGB frame, which adds extra Blender post-processing effects to the scene; and (4) a body parts' segmentation frame. Moreover, the 2D and 3D body pose of each human model is generated and exported to JavaScript Object Notation (JSON) format.

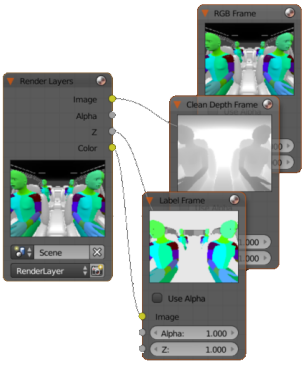


Figure 5.5: Blender internal rendering nodes. (Top to Bottom): RGB, Depth, Label.

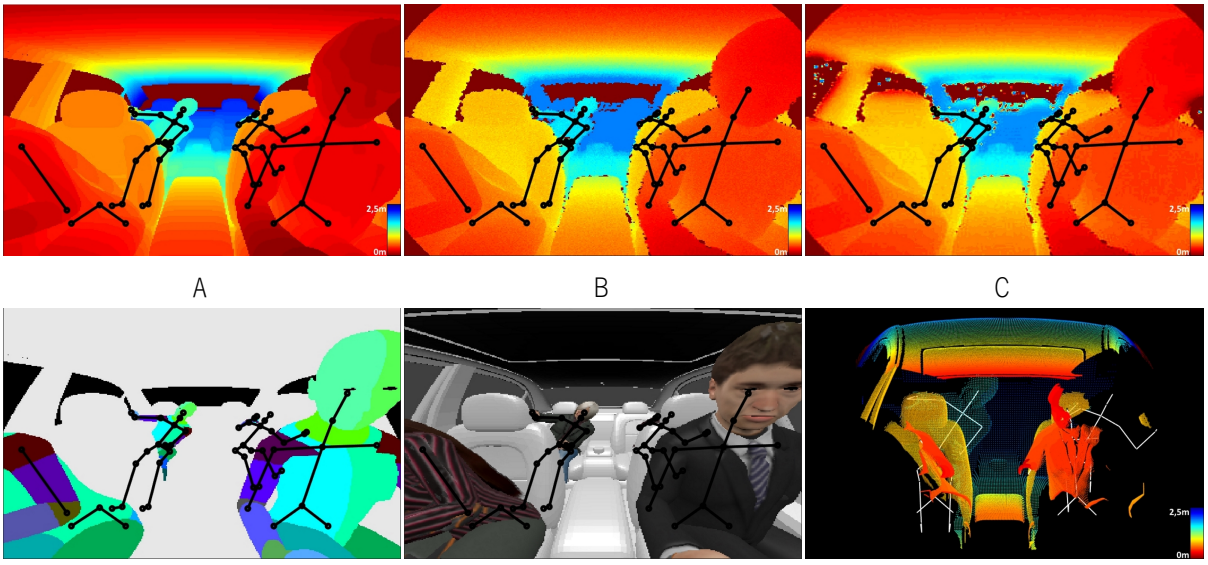


Figure 5.6: Rendered frames: (A) depth, (B) noise, (C) NST, (D) labels, (E) RGB and (F) point cloud. Depth images (A, B, C) are represented in color for better visualization. Dots/lines represents the ground-truth (black -> 2D, white -> 3D).

5.1.3.1 Clean depth frame

To render a depth frame (Figure 5.6A), $\boldsymbol{depth_{x,y}}$, the toolchain needs to extract the z-channel of the RGB sensor in Blender through the "RenderLayers" node (Figure 5.5). This z-channel provides a matrix with the same size as the RGB sensor, where each pixel information is the distance from the object projected on the pixel in the camera $\boldsymbol{XY}$ plane.

5.1.3.2 Noise depth frame

The noise frame (Figure 5.6B), is generated using several post-processing effects.
A 2^{nd} degree equation model (equation 5.5), used to generate a Gaussian value for each pixel of the frame, $\boldsymbol{noisedepth_{x,y}}$. The noise model only relates the axial noise with the depth component, and it does not consider the angle between the surface normal and the camera's axes [80].

$$\begin{aligned} \sigma_{x,y} &= \Gamma^a + \Gamma^b \times depth_{x,y} + \Gamma^c \times depth_{x,y}^2 \\ noisedepth_{x,y} &= Gaussian(\sigma \leftarrow \sigma_{x,y}, \mu \leftarrow depth_{x,y}) \end{aligned} \tag{5.5}$$

Besides general noise modelling, other frame aspects were mimicked. In this sense, specific Gaussian noise is added to abrupt depth edges (identified using an empirical threshold, $\boldsymbol{th}$, and kernel, $\boldsymbol{k,w}$, applied to an edge image computed by central finite differences [equation 5.6]).

$$noisedepth_{x,y} = \begin{cases} 0, noisedepth_{x,y} < noisedepth_{x\pm k,y\pm w} - th \\ 0, noisedepth_{x,y} > noisedepth_{x\pm k,y\pm w} + th \\ noisedepth_{x,y}, \end{cases} \tag{5.6}$$

A circular crop (equation 5.7) is added to simulate the real ToF images FoV, where $\boldsymbol{x_c}$,$\boldsymbol{y_c}$,$\boldsymbol{ra}$ represent the circumference center $\boldsymbol{xy}$ pixel coordinate and radius.

$$noisedepth_{x,y} = \begin{cases} noisedepth_{x,y}, ((x - x_c)^2 + (y - y_c)^2) < ra^2 \\ 0, ((x - x_c)^2 + (y - y_c)^2) \geq ra^2 \end{cases} \tag{5.7}$$

Finally, a dead-zone, $\boldsymbol{dz}$, and saturation, $\boldsymbol{sat}$, simulates the real ToF images range (equation 5.8).

$$noisedepth_{x,y} = \begin{cases} noisedepth_{x,y}, dz < noisedepth_{x,y} < sat \\ 0, dz > noisedepth_{x,y} \cup noisedepth_{x,y} > sat \end{cases} \tag{5.8}$$

5.1.3.3 NST depth frame

The NST frame (Figure 5.6C) is generated with a NST method [106]. The toolchain generates a new NST frame, $\boldsymbol{nstdepth_{x,y}}$, for each synthetic frame, by feeding the network with a real ToF image (Chapter 4) serving as style, $\boldsymbol{x_S}$, (Figure 5.7B), and the synthetic frame, $\boldsymbol{noisedepth_{x,y}}$, serving as content, $\boldsymbol{x_C}$, (Figure 5.7A). The strategy aims to better infer the noise style and add it into the generated synthetic frames (Figure 5.7C).

A B C

Figure 5.7: NST frames: (A) content, (B) style, and (C) resulting NST-based synthetic image.

5.1.3.4 Labels frame

To render a labels frame (Figure 5.6D), $\boldsymbol{labels_{x,y}}$, $\boldsymbol{x = 1, ..., X}$, $\boldsymbol{y = 1, ..., Y}$, $\mathbb{N} \cap [0x000000, 0xFFFFFF]$, the toolchain needs to extract the color-channel of the RGB sensor in Blender

through the "RenderLayers" node (Figure 5.5). This channel gives a matrix with the size of the RGB sensor, where each pixel information is the RGB color code from the object projected on the pixel. It is important to use this channel, because it does not use post-processing effects when rendered, such as ray-tracing, ambient occlusion, ambient light or shadowing, giving a raw RGB code of the projected texture, preserving the human models' segmented skin, $\boldsymbol{\Gamma_h^{lb}}$ (Figure 5.2).

5.1.3.5 RGB frame

Similar to the labels frame, the RGB frame (Figure 5.6E), $\boldsymbol{RGB_{x,y}}$, $\boldsymbol{x = 1, ..., X}$, $\boldsymbol{y = 1, ..., Y}$, $\mathbb{N} \cap [0x000000, 0xFFFFFF]$ captures the RGB sensor rendering, but in this case it adds all the post-processing effects, in order to improve the scene realism. To capture this information, we need to use the image-channel of the RGB sensor in Blender through the "RenderLayers" node (Figure 5.5). Note that, the toolchain automatically switches the human models skin texture, $\boldsymbol{\eta_h^{st}}$, with a realistic one, $\boldsymbol{\Gamma_h^{rgb}}$, before rendering. This skin is pre-selected by the user for each human model.

5.1.3.6 Point cloud

The 3D point cloud, $\boldsymbol{pc_f}$, (Figure 5.6F) has the Cartesian coordinates $\boldsymbol{xyz}$ of the voxel that was projected in each pixel, $\boldsymbol{pcx_f, pcy_f, pcz_f}$, $\boldsymbol{f = 1, ..., X \times Y}$. As mentioned before, Blender does not give this information in a simple and straightforward way. The toolchain relies on the information of the NST depth frame, $\boldsymbol{nstdepth_{x,y}}$, the camera's resolution $\boldsymbol{\iota^X}$ and $\boldsymbol{\iota^Y}$, and its horizontal FoV, $\boldsymbol{\iota^{HFoV}}$, in order to calculate each voxel position with respect to its pixel projection (equation 5.9), as illustrated in Figure 5.8.

$$
\begin{aligned}
pixel2degree &= \frac{\tan(\frac{\iota^{HFoV}}{2})}{\left(\frac{\iota^X}{2}\right)} \\
pcx_f &= nstdepth_{x,y} \times \left(x - \frac{\iota^X}{2}\right) \times pixel2degree \\
pcy_f &= nstdepth_{x,y} \times \left(y - \frac{\iota^Y}{2}\right) \times pixel2degree \\
pcz_f &= nstdepth_{x,y} \\
\therefore pc_f &= \{pcx_f, pcy_f, pcz_f\}
\end{aligned}
\tag{5.9}
$$

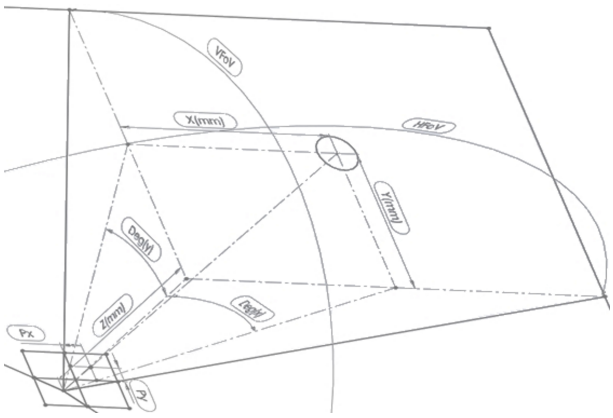
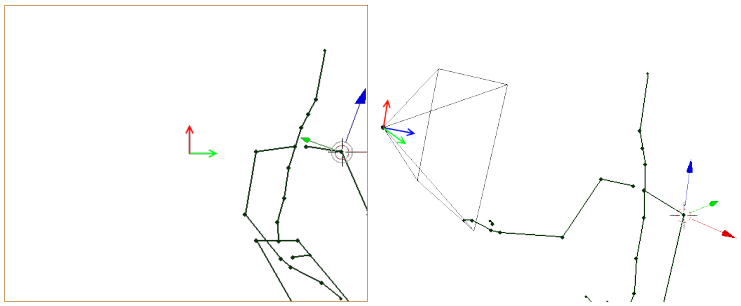


Figure 5.8: ToF 2.5D depth pixel to 3D point-cloud pixel.

5.1.3.7 Ground-Truth

The ground-truth information (black skeleton lines in Figure 5.6) consists in exporting the pose information for each human model, with respect to the camera, much like in the real dataset generation toolchain (Chapter 4). With that in mind, we have 2 types of ground-truth (Figure 5.9): 2D pose for depth $(\boldsymbol{Pxy}_{\boldsymbol{J}}^{\boldsymbol{C}})_{\boldsymbol{h}}$, labels and RGB frames; and 3D pose for the point cloud frame $(\boldsymbol{Pxyz}_{\boldsymbol{J}}^{\boldsymbol{C}})_{\boldsymbol{h}}$, where $\boldsymbol{h}$ represents the human index for the body pose. Both types of ground-truth consist in the same pose information for each human model, that is a structure comprised of all joints' pixel (2D) or voxel (3D) positions. This structure is directly related to the MakeHuman human model skeleton (Table 5.3). When generating both 2D and 3D ground-truth, the toolchain automatically does the necessary transformations from the global coordinate system to the camera's local coordinate system. This transformation is done for each joint in Table 5.3, and its ground-truth projection can be seen for each human in each frame, as illustrated in Figure 5.6.



A B

Figure 5.9: Ground-truth perspective wrt. ToF image sensor: (A) 2D lshoulder global coordinates to camera 2D pixel coordinates, and (B) 3D lshoulder global coordinates to camera local coordinates.

Table 5.3: Body pose joint label correspondence between ground-truth (Table 4.1) and MakeHuman human model.

Ground-Truth	MakeHuman	Ground-Truth	MakeHuman
head	Head	rhand	RightFingerBase
neck	Neck	hip	Hips
chest	Spine1	lpelvis	LeftUpLeg
lshoulder	LeftArm	rpelvis	RightUpLeg
rshoulder	RightArm	lknee	LeftLeg
lelbow	LeftForeArm	rknee	RightLeg
relbow	RightForeArm	lankle	LeftFoot
lwrist	LeftHand	lfoot	LeftToeBase
rwrist	RightHand	rankle	RightFoot
lhand	LeftFingerBase	rfoot	RightToeBase

5.2 Implementation Details

Considering the presented methodology, several implementation considerations were made. Although the toolchain generates data for the mentioned sensors, it does not try to achieve full realism for them, only adding axial, edge, circular crop, range and stylized noise. The focus is placed in guaranteeing that the data output is of the same format, putting the biggest effort in the automation of the data generation procedure, as well as in the validation of all generated data. The toolchain is an interaction between the Graphical User Interface (GUI) and the toolchain's internal engines, generating data using three separate engines.

Blender was the selected main engine for the toolchain, given the easy access to 3D customizable environments, camera customization and rendering, customizable human models interaction, and most importantly a good interface with Python. This interface also gives an easy access to all Blender embedded functions, so that all manual user interactions with Blender can be made programmatically. The Central Processing Unit (CPU) resource's allocation for each toolchain instance is managed by Blender in the following manner: one CPU core for the Python engine and the remaining ones for specific Blender rendering functions.

In order to create human models automatically, the MakeHuman engine was selected and embedded into the toolchain. With its easy Python interface, the toolchain is able to programmatically access its functions and generate random human models.

The last but not least was the the Python engine, used for the purpose of toolchain automation. Due to its complex implementation in-depth discussion is presented in Section 5.2.1.

5.2.1 Python Engine

The second most important engine is the Python engine. This engine is responsible for all the toolchain automation. At first, it calls and waits for the GUI to finish loading up all the user settings. After this initial process, it starts constantly interacting with Blender, while generating datasets. With it, it is possible to control the entire pipeline, from user-specific customization, to scene initialization, generation, validation, and finally the output rendering.

5.2.1.1 Toolchain automation

In Algorithm 5.2.1, it is possible to understand the entire cycle of automation. The inputs from the algorithm are the outputs from the GUI, considering the camera, humans and car customization.

Algorithm 5.2.1 Python Engine

inputs:
  # Get user input
  Γ^{T}
  Γ^{C}
  Γ^{H}_{h}
  Γ^{NF}
initialize:
  # Create camera model
  $\iota \leftarrow \Gamma^{T}$
  # Create car model
  $\zeta \leftarrow \Gamma^{C}$
  # Define initial pose for each human
  $\eta^{bp}_{h,j,a} \leftarrow \Gamma^{ip}_{h,j,a}$
  # Create 3d skeleton for each human
  $\eta^{sk}_{h} \leftarrow new3dskeleton(\eta^{bp}_{h,j,a})$
while $t' < \Gamma^{NF}$ **do**
  Alg. 5.2.2($\eta^{bp}_{h,j,a}$, $\Gamma^{g}_{h,j,a}$**)**
  $\eta^{sm}_{h} \leftarrow \Gamma^{lb}_{h}$
  $BlenderUpdateScene()$
  $collision \leftarrow$**Alg. 5.2.4(**η^{sm}_{h}, η^{sk}_{h}**)**
  if $collision = false$ **then**
    $collision \leftarrow$**Alg. 5.2.5(**η^{sm}_{h}**)**
    if $collision = false$ **then**
      $collision \leftarrow$**Alg. 5.2.6(**η^{sm}_{h}, ζ^{m}**)**
      if $collision = false$ **then**
        $t' \leftarrow t' + 1$
        $depth_{x,y}$ (**Figure 5.5**)
        $labels_{x,y}$ (**Figure 5.5**)
        $noisedepth_{x,y} \leftarrow$**Alg. 5.2.7(**$depth_{x,y}$**)**
        $nstdepth_{x,y} \leftarrow$**Alg. 5.2.8(**$noisedepth_{x,y}$,$labels_{x,y}$**)**
        $pc_{f} \leftarrow$**Alg. 5.2.9(**$depth_{x,y}$, ι**)**
        $\eta^{sm}_{h} \leftarrow \Gamma^{rgb}_{h}$
        $BlenderUpdateScene()$
        $RGB_{x,y}$ (**Figure 5.5**)
        for each h in H **do**
          $(Pxy^{C}_{J})_{h} \leftarrow$**Alg. 5.2.10(**$\eta^{bp}_{h,j,a}$ ι**)**
          $(Pxyz^{C}_{J})_{h} \leftarrow$**Alg. 5.2.11(**$\eta^{bp}_{h,j,a}$, ι**)**
        end for
      end if
    end if
  end if
end while

The initialization procedure focuses in loading the GUI settings into Blender environment/variables, in order to create the 3D scenario that was intended for data generation. An extra step is added to this procedure, and it consists in the creation of the 3D skeleton for each human, assisting in the body to body collision detection (Figure 5.4A). After these two procedures, the automated cycle starts, until all valid frames, $\boldsymbol{t'}$, have been rendered. For each rendered frame, the toolchain generates a new pose for each human model, checks the 3 types of collisions for each human, and if there are no collisions, then it starts rendering the data frames and ground-truth, while making the proper skin mesh changes for specific image frames. Much like the real dataset, $\boldsymbol{RD_{t'}}$, in equation 4.13, the synthetic dataset follows the same dataset standard, with the added frames and per human ground-truth information. Each exported dataset, $\boldsymbol{SD_{t'}}$, is comprised by the group of dataset information (equation 5.10).

$$\begin{aligned} \therefore SD_{t'} = \{&depth_{t',x,y}, labels_{t',x,y}, noisedepth_{t',x,y}, nstdepth_{t',x,y}, RGB_{t',x,y}, \\ &pc_{t',f} = \{pcx_{t',f}, pcy_{t',f}, pcz_{t',f}\}, \\ &(Pxy_J^C)_{t',h}, (Pxyz_J^C)_{t',h}\} \end{aligned} \tag{5.10}$$

5.2.1.2 Body pose generation

The algorithm for Gaussian body pose generation (Algorithm 5.2.2) uses the information that each human model has with respect to its joints' freedom of movement, and the correspondent axis Gaussian/incremental distribution, in order to generate new angular values for each axis joint. For each human, it cycles through its joints' and axes, generating random Gaussian/incremental angles. We can consider this algorithm complexity for the worst case as $\boldsymbol{\mathcal{O}(H \times Jf \times Af)}$, where $\boldsymbol{H}$, $\boldsymbol{Jf}$ and $\boldsymbol{Af}$ are the number of human models, joints' in the human model, and degrees of freedom in the joint.

Algorithm 5.2.2 Gaussian Body Pose Generation

inputs:
$\eta^{bp}_{h,j,a}$
$\Gamma^{g}_{h,j,a}$
for each h in H **do**
  for each j in Jf **do**
    for each a in Af **do**
      $\eta^{o}_{h,j,a} \leftarrow Gaussian(\mu \leftarrow \Gamma^{\mu}_{h,j,a}, \sigma \leftarrow \Gamma^{\sigma}_{h,j,a})$
    end for
  end for
end for

5.2.1.3 Collision detection

As mentioned before, there are 3 types of collision detection built into the toolchain, that guarantee the quality of the generated dataset. All detections are based on the principle of "overlapping meshes" (Algorithm 5.2.3, and others therein).

If we look into Algorithm 5.2.1, we can see that the collision detection algorithm is called sequentially for the body-body, human-human and human-car detections, being its performance for the worst case given by equation 5.11. Note that the implementation allows for improved performance, as collisions are only evaluated when the previous ones are validated, thus avoiding unnecessary evaluation bottlenecks.

Algorithm 5.2.3 Collision Detection

inputs:
$Mesh_1$
$Mesh_2$
$bvhtree1 \leftarrow FromBMesh(Mesh_1)$
$bvhtree2 \leftarrow FromBMesh(Mesh_2)$
$intersected \leftarrow bvhtree1.overlap(bvhtree2)$

Algorithm 5.2.4 Body To Body Collisions

inputs:
η_h^{sm}
η_h^{sk}
for each h in H **do**
  if $Alg.\ 5.2.3(\eta_h^{sm}, \eta_h^{sk})$ **then**
    return $true$
  end if
end for
return $false$

Algorithm 5.2.5 Human To Human Collisions

inputs:
η_h^{sm}
for each h in $H - 1$ **do**
  for $u \leftarrow h + 1$ to H **do**
    if $Alg.\ 5.2.3(\eta_h^{sm}, \eta_u^{sm})$ **then**
      return $true$
    end if
  end for
end for
return $false$

Algorithm 5.2.6 Human To Car Collisions

inputs:
η_h^{sm}
ζ^m
for each h in H **do**
  if $Alg.\ 5.2.3(\eta_h^{sm}, \zeta^m)$ **then**
    return $true$
  end if
end for
return $false$

$$
\begin{aligned}
body \longleftrightarrow body &= \mathcal{O}(H) \\
human \longleftrightarrow human &= \mathcal{O}\left(\frac{H^2 + H}{2}\right) \\
human \longleftrightarrow car &= \mathcal{O}(H) \\
\therefore collision &= \mathcal{O}(H^2), H \longrightarrow +\infty
\end{aligned}
\tag{5.11}
$$

5.2.1.4 Noise Model

In order to use a realistic noise model, we performed a noise regression to a specific ToF camera (Pico Monstar 105). The method consisted in placing the camera in front of a white wall, at $\boldsymbol{D}$ different distances, and recording $\boldsymbol{Fr}$ frames for each distance (5 and 100 respectively in our experiments). The standard deviation of the error is then calculated for each distance $\boldsymbol{\sigma_d}$, $\boldsymbol{d = 1, ..., D}$ (equation 5.12), finally estimating a regression for the model $\boldsymbol{\Gamma^n}$. For the specific case of the ToF camera used, $\boldsymbol{\Gamma^a = 1}$, $\boldsymbol{\Gamma^b = 5}$ and $\boldsymbol{\Gamma^c = 1}$ as illustrated in Figure 5.10.

$$
\begin{aligned}
\mu_{d,x,y} &= \frac{\sum_{f=1}^{Fr} depth_{d,x,y,f}}{Fr} \\
\sigma_{d,x,y} &= \sqrt{\frac{\sum_{f=1}^{Fr} (depth_{d,x,y,f} - \mu_{d,x,y})^2}{Fr - 1}} \\
\sigma_d &= \frac{\sum_{x=1}^{X} \sum_{y=1}^{Y} \sigma_{d,x,y}}{X \times Y}
\end{aligned}
\tag{5.12}
$$

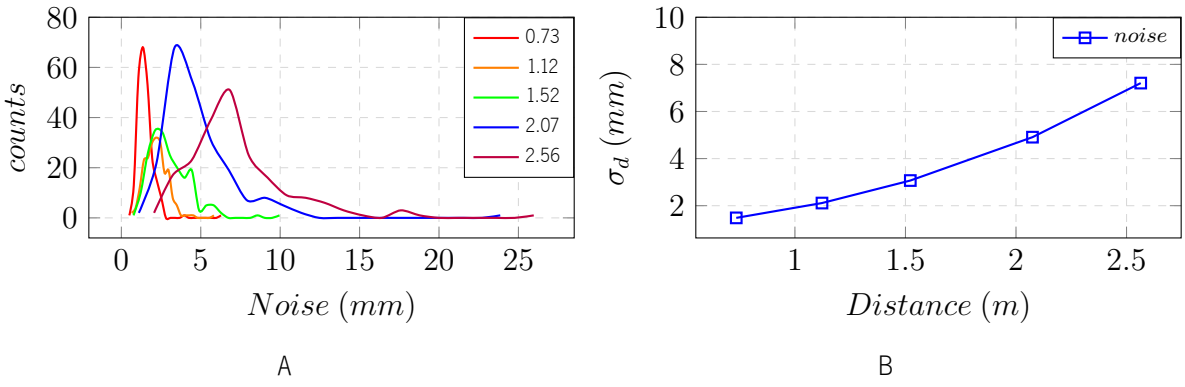


Figure 5.10: Pico Monstar 105 gaussian noise: (A) gaussian noise $\boldsymbol{\sigma_{d,x,y}}$ for each distance value in plot (B); (B) gaussian noise regression, $\boldsymbol{noise \equiv \Gamma^n = \Gamma^a + \Gamma^b \cdot z + \Gamma^c \cdot z^2}$.

Algorithm 5.2.7 shows the complete noise model added to the $\boldsymbol{noisedepth_{x,y}}$ frame. Circular crop (equation 5.7) and edge Gaussian noise were empirically set, contrary to noise regression (equation 5.12) , dead zone and saturation that were set with Pico Monstar 105 Use Case 3 in Table 2.3, representing the same settings from the real dataset generation toolchain (Chapter 4).

Algorithm 5.2.7 Noise Frame

1: **inputs:**
$depth_{x,y}$
2: $noisedepth_{x,y} \leftarrow Eq.\ 5.5(depth_{x,y}, \Gamma^n)$
3: $noisedepth_{x,y} \leftarrow Eq.\ 5.6(noisedepth_{x,y})$
4: $noisedepth_{x,y} \leftarrow Eq.\ 5.7(noisedepth_{x,y})$
5: $noisedepth_{x,y} \leftarrow Eq.\ 5.8(noisedepth_{x,y}, \{dz, sat\} \leftarrow UseCase3inTable\ 2.3)$

5.2.1.5 NST Model

To to improve the noise model, a noise stylizing [106] is added on top of the noise model. Spatial control is customized (Figure 5.11), where guidance channels, $\boldsymbol{T_l^r}$, are created for each of the VGG19 Convolutional Neural Network (CNN) layers, $\boldsymbol{l}$, each being a binary mask image map of values in $\boldsymbol{[0,1]}$. Three spatial regions, $\boldsymbol{r}$, were created (equation 5.13 shows the procedure for content frames, $\boldsymbol{x_C}$ [Figure 5.7A]) for windows, $\boldsymbol{T_l^o}$, car, $\boldsymbol{T_l^o}$, and human, $\boldsymbol{T_l^o}$.

$$
\begin{aligned}
\therefore T_l^o &= \begin{cases} 1, labels_{x,y} \cap Table\ 5.1 \\ 0, \end{cases} \\
\therefore T_l^o &= \begin{cases} 1, labels_{x,y} = 0xE7E7E7 \\ 0, \end{cases} \\
\therefore T_l^o &= 1 - (T_l^o + T_l^o)
\end{aligned}
\tag{5.13}
$$

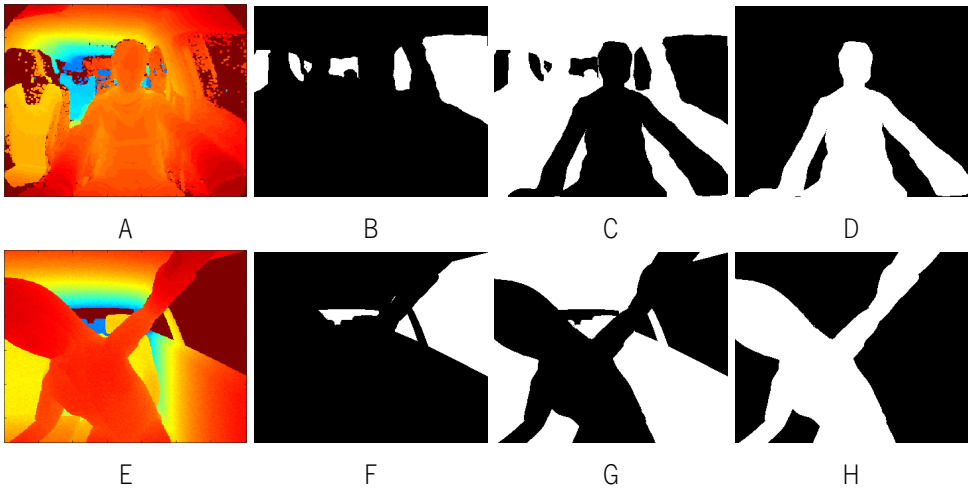


Figure 5.11: NST spatial control guidance channels frames for input frame: (A to D) Real depth frame, $\boldsymbol{x_S}$, and its $\boldsymbol{T^0}$, $\boldsymbol{T^1}$ and $\boldsymbol{T^2}$ guidance channels, and (E to H) Synthetic depth frame, $\boldsymbol{x_C}$, and its $\boldsymbol{T^0}$, $\boldsymbol{T^1}$ and $\boldsymbol{T^2}$ guidance channels.

Through empirical selection, VGG19 layers "conv2_1" and "conv3_1" were used, $\boldsymbol{clayers}$, for image content, $\boldsymbol{x_C}$, and layers "conv2_2", "conv3_4", and "conv4_4" has style layers, $\boldsymbol{slayers}$, as well as VGG19 pre-trained weights, $\boldsymbol{weights}$. This information is used for the $\boldsymbol{nstdepth_{x,y}}$ creation as shown in Algorithm 5.2.8.

Algorithm 5.2.8 NST Frame

inputs:
$noisedepth_{x,y}$
$labels_{x,y}$
initialize:
$weights \leftarrow VGG19$
$clayers \leftarrow \{conv2\_1, conv3\_1\}$
$slayers \leftarrow \{conv2\_2, conv3\_4, conv4\_4\}$
$\{T_l^o, T_l^o, T_l^o\} \leftarrow Eq.\ 5.13(labels_{x,y})$
$nstdepth_{x,y} \leftarrow NST(x_C \leftarrow noisedepth_{x,y}, x_S \leftarrow depth_{t',x,y}$ **Eq. 4.13)** [106]

5.2.1.6 Point cloud

Point cloud generation is a simple algorithm (Algorithm 5.2.9) that makes use of equation 5.9, both for initialization, as well as to calculate the entire voxel projection of each pixel in the $\boldsymbol{nstdepth_{x,y}}$ frame. We can consider its performance for the worst case as $\boldsymbol{\mathcal{O}(X \times Y)}$, $\boldsymbol{x = 1, ..., X}$, $\boldsymbol{y = 1, ..., Y}$, where $\boldsymbol{X}$ is the camera's horizontal resolution and $\boldsymbol{Y}$ its vertical resolution.

Algorithm 5.2.9 Point cloud

inputs:
  $nstdepth_{x,y}$
  ι
initialize:
  $pixel2degree$ **in Eq.5.9(**ι^{HFoV}, ι^{X}**)**
$f \leftarrow 0$
for each x in ι^{X} **do**
  for each y in ι^{Y} **do**
    pc_f **in Eq.5.9(**$nstdepth_{x,y}$**, pixel2degree, f)**
    $f \leftarrow f + 1$
  end for
end for

5.2.1.7 Ground-truth

Two algorithms were implemented (algorithms 5.2.10 and 5.2.11) for easy transformation from each joint global coordinates and axes to the cameras local coordinates and axes.

Algorithm 5.2.10 2D Ground-Truth

inputs:
  $\eta^{bp}_{h,j,a}$
  ι
for each j in Jf **do**
  $jointtranslation \leftarrow \eta^{bp}_{h,j,a}.xyz.to\_translation()$
  $projected \leftarrow bpy\_extras.object\_utils.world\_to\_camera\_view($
    $bpy.context.scene,$
    $\iota.xyz,$
    $jointtranslation)$
  $s \leftarrow bpy.context.scene.render.resolution\_percentage/100$
  $x \leftarrow projected.x \times bpy.context.scene.render.resolution\_x \times s$
  $y \leftarrow projected.y \times bpy.context.scene.render.resolution\_y \times s$
  $(Pxy^{C}_{J})_h = \{x, y\}$
end for

Algorithm 5.2.11 3D Ground-Truth

inputs:
  $\eta^{bp}_{h,j,a}$
  ι
for each j in Jf **do**
  $jointtranslation \leftarrow \eta^{bp}_{h,j,a}.xyz.to\_translation()$
  $projected \leftarrow (jointtranslation - \iota.xyz) \times \iota.matrix\_world$
  $x \leftarrow projected.x$
  $y \leftarrow projected.y$
  $z \leftarrow projected.z$
  $(Pxyz^{C}_{J})_h = \{x, y, z\}$
end for

5.3 Evaluation results

5.3.1 Performance

Given that Blender interacts with Python in a very specific way, the toolchain is limited when trying to use CPU multitasking inside the Python engine. Each time a toolchain instance is called, threads are allocated to the Blender engine for embedded functions and rendering, leaving no room or access to Python multitasking functions. Also, Graphics Processing Unit (GPU) usage is currently of limited access, with the toolchain being executed without 3D visible environment as it is constrained for specific Blender rendering functions. With all these considerations, we performed one type of performance evaluation to better understand how to improve the synthetic data generation throughput, considering the performance of each algorithm. The evaluation consists in changing the number of human models and the number of toolchain parallel instances. All the evaluations were performed on a computer with a CPU: i7-7700HQ @3.56GHz, Random-Access Memory (RAM): 16GB DDR4 @1.2GHz, GPU: GTX1070 @1.443GHz, 8GBytes GGDR5 @2GHz, Storage: Solid-State Drive (SSD) NVMe @1.693GB/s (Read) @0.869GB/s (Write), OS: Windows 10 Education x64.

5.3.1.1 CPU load

From the results, it was possible to understand that CPU usage is highly dependent on the number of parallel instances being executed, and not on the number of human models being used (Figure 5.12A). This corroborates the idea that Blender limits the Python engine with single thread access, something that was visible when trying to multi-task with specific Python instructions. With the execution of parallel instances, the operative system can give more thread access to the entire toolchain synthetic data generation.

5.3.1.2 Processing time

Processing time can be divided in two main sequences in Algorithm 5.2.1, collisions (body-body, human-human and human-car) and rendering. Collision detection performance is highly dependent on the number of human models (equation 5.11), while rendering is mostly affected by frame resolution $\mathcal{O}(X \times Y)$, number of human models $\mathcal{O}(H)$, write latency caused by disk concurrent access and CPU usage when each instance uses Blender internal render, creating a 100% CPU load spike. In Figure 5.12B, we notice that with the increase in instances, rendering time increased the most, mostly due to disk concurrent access and the reduced CPU resources. Collision detection was the least affected, being affected only on the CPU side. When comparing the increase in human models for each number of instances, it is possible to see that the rendering time

suffers the least increase, being related with the increase in gound-truth files being generated (one for each human model). In the case of collision detection, the processing time increases significantly, being related to the quadratic behaviour in equation 5.11.

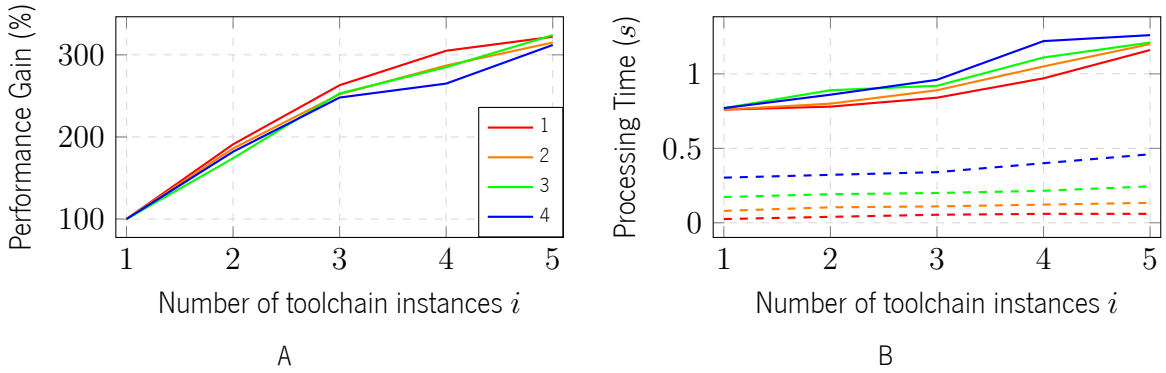


Figure 5.12: Toolchain performance when increasing the number of instances $\boldsymbol{i}$, for different number of humans $\boldsymbol{h}$: (A) total gain $\boldsymbol{gain_{h,i}}$; (B) single instance processing time, where $dashed \rightarrow Collision, solid \rightarrow Rendering$.

5.3.1.3 Toolchain performance

In terms of total toolchain performance, the results show that increasing the number of human models does reduce the performance for single and multi-instances, and this understanding comes from the previous analysis on specific algorithms' performance. However, total toolchain performance achieves higher gains when increasing the number of parallel instances, starting to saturate when reaching full CPU load, as illustrated in Figure 5.12A. In terms of absolute performance, the best scenario was achieved when running 5 instances with 1 human model, generating synthetic data at 4.1Hz.

5.3.2 Application to Pose Estimation Problems

To understand the validity of the data being generated with our toolchain, as well as its ability to increase ML algorithmic accuracy, we defined four distinct experimental scenarios: 2D pose estimation from depth images; 2D pose estimation from point cloud; 3D pose estimation from 2D pose; human body parts segmentation from depth images.

5.3.2.1 Evaluation Data

To provide the experimental scenarios with valid datasets that allow us to evaluate the advantage of combining synthetic and real data, we require both real and synthetic samples. In this sense, we used a publically available dataset MoLa R10k InCar Dataset [108] (generated with the toolchain presented in

Chapter 4, plus the synthetic dataset generated by the proposed toolchain). The real dataset consists in three recorded subjects, $S_{1:3}$, two redundant actions, $A_{1:2}$, each, totaling 10482 samples. In its turn, the synthetic dataset comprises data generated using seven car models, $CM_{1:7}$, and eighteen subjects, $Z_{1:18}$, with associated Gaussian poses, N_{hgp}, totalling 25200 samples. Both datasets are identical in terms of sample data types for depth frame (i.e. NST depth frame for the synthetic dataset), point cloud, 2D and 3D body pose, giving us the opportunity to evaluate the three first experimental scenarios in a quantitative way. Lack of real in-car body parts' segmentation frames in publically available datasets led us to define a qualitative evaluation for the fourth experiment. The available samples were divided into 3 groups: (1) a training set, with all synthetic samples plus 6946 real samples (corresponding to subjects S_1 and S_2); (2) a validation set with 900 real samples; and (3) a test set with 2636 real samples. Groups (2) and (3) use samples from subject S_3 performing distinct actions. To assess the influence of the ratio between real and synthetic images and the influence of the number of real images available, we trained each network with different amounts of samples, establishing ten sub-evaluations ($R_{1:10}$) for each of the first three experiments (Table 5.4).

Table 5.4: Evaluations related with real and synthetic data quantities/ratios. Each $R\#$ represents a sub-evaluation for the assessment of the influence of mixing real and synthetic datasets.

Evaluation	Real	Synthetic	Total	Ratio
R1	900	0	900	1:0
R2	900	2700	3600	1:3
R3	900	4500	5400	1:5
R4	900	9000	9900	1:10
R5	1800	0	1800	2:0
R6	1800	5400	7200	2:6
R7	1800	9000	10800	2:10
R8	1800	18000	19800	2:20
R9	6946	0	6946	7.7:0
R10	6946	25200	25200	7.7:28

To account for the stochastic nature of the training, each sub-evaluation was repeated 3 times, using a different set of samples, with the metrics being averaged over the 3 trained models. For $R_{8:10}$, due to lack of new samples for the different folds, the training was repeated thrice upon shuffling the samples. The proposed synthetic dataset, plus the necessary tools, to reproduce all experiments were made publically available [109] (Figure 5.13).

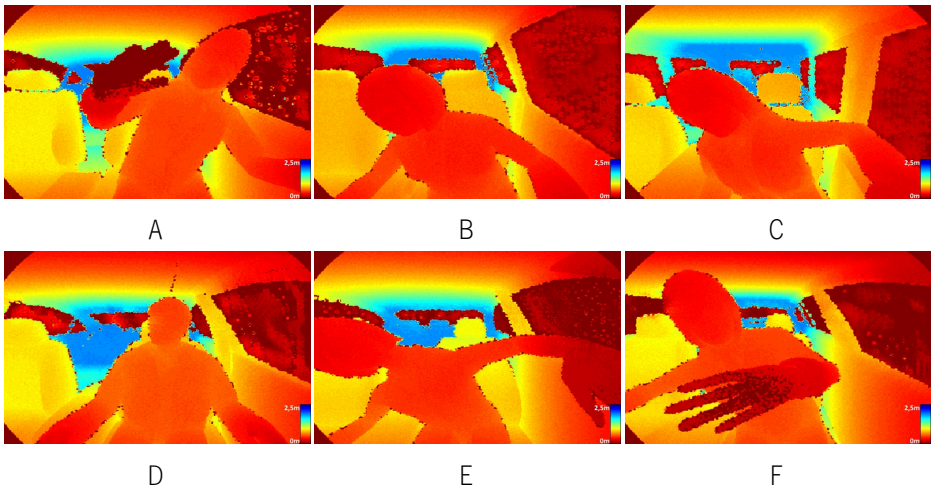


Figure 5.13: Synthetic frames: (A) Car A with male, (B) Car B with female, (C) Car C with female, (D) Car D with male, (E) Car E with female and (F) Car F with male. Depth images (A to F) are represented in color for better visualization.

5.3.2.2 2D Pose Estimation from Depth Images (SE1)

To evaluate the synthetically generated depth frames and corresponding 2D ground-truth, the Part Affinity Fields (PAF) [25] method was used. From it, a custom CNN was implemented consisting only on the first stage of the original PAF CNN. In each sub-evaluation, ***R#***, the method used the depth frame as input features (raw depth for real dataset, NST depth for synthetic dataset) and the 2D body pose as output labels (Figure 5.14). For all samples, the depth frame was normalized into a grayscale frame ($[0; 1.8]\ m \equiv [0; 255]$), while each 2D joint position was converted into a 2D heatmap. For metric evaluation, the joint position is estimated through non-maximum suppression applied to the inferred heatmap. In this experiment, the PCKh measure (in pixels, using a matching threshold given by 50% of the head segment length) and the Area Under Curve (AUC) were used as metrics [103]. Table 5.5 summarizes the average results for the full body, with the results for individual joints being presented in Table 5.6. Figures 5.15A and 5.16 present the PCKh@0.5 values for the full body and body parts for each sub-evaluation respectively.

5.3.2.3 2D Pose Estimation from Point cloud (SE2)

To evaluate the synthetically generated point cloud and corresponding 2D ground-truth, the PAF [25] method was used. In this experiment, the point cloud was used as input features (raw point cloud for real samples and NST-based point cloud for synthetic samples). To this end, each point cloud was normalized (***pcx*** and ***pcy*** with $[-1.5; 1.5]\ m \equiv [0; 255]$, and ***pcz*** with $[0; 1.8]\ m \equiv [0; 255]$) and converted into a 3-channel matrix. As for SE1, the network's output was the 2D heatmaps generated from each joint's

position, with the inferred joint position being computed by non-maximum suppression. The same metrics from SE1 were employed. Results are shown in Tables 5.5, 5.6, and Figures 5.15B and 5.16.

5.3.2.4 3D Pose Estimation from 2D Pose (SE3)

To evaluate the synthetically generated 3D ground-truth, a 3D pose estimation method [104] was used. The method uses a 2D body pose as input features (provided as joint pixel coordinates) and the 3D body pose as output (Figure 5.14). Once again, similar metrics were employed, but in this case PCKh matching threshold was normalized to a fixed head size of 200 mm. Results are shown in Tables 5.5, 5.6, and Figures 5.15C and 5.16.

5.3.2.5 Human Body Segmentation from Depth Images (SE4)

To evaluate the synthetically generated point cloud and corresponding segmentation frames, the Convolutional Networks for Biomedical Image Segmentation (U-Net) [110] method was used (Figure 5.17). The method was implemented in two stages, where the first stage, $\boldsymbol{St_1}$, infers the human silhuete from the background, to mask the input image features that are then used on the second stage, $\boldsymbol{St_2}$, to infer the human body parts. The method uses the NST point cloud as input features and the labels frame as output labels (Figure 5.14). For all samples, the point cloud was normalized and converted into a 3-channel matrix, while each labels frame was converted into a one-hot representation with 8 body parts (each body part's RGB code was converted into an intensity value [0;N-1], where $\boldsymbol{N}$ represents the body part index). Evaluation is done qualitatively and results are shown in Figure 5.18 (upon inferring the class [label] with maximum probability is computed per pixel, generating the final predicted labels frame).

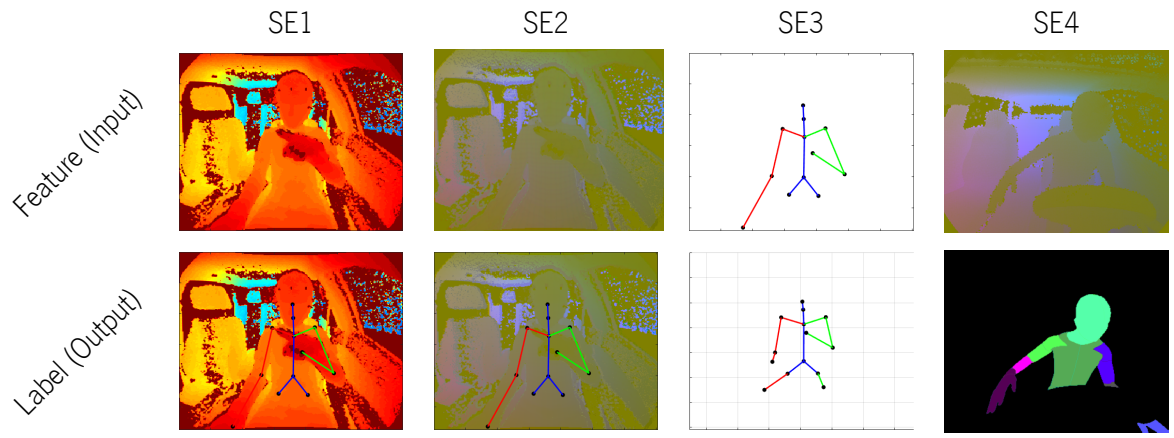


Figure 5.14: Visual representation of input features and output used for each experimental scenario $SE\#$: (SE1) 2D pose estimation from depth images using normalized depth frame as input and 2D body pose as output; (SE2) 2D pose estimation from point cloud using normalized point cloud as input and 2D body pose as output; (SE3) 3D pose estimation from 2D pose using 2D body pose as input and 3D body pose as output; and (SE4) human body parts segmentation from point cloud images using normalized point cloud as input and segmentation frame as output.

Table 5.5: PCKh measure and AUC values averaged over all 14 joints, for the 3 experimental scenarios and all 10 sub-evaluations. Each ***R#*** represents a sub-evaluation for the assessment of the influence of mixing real and synthetic datasets. Each ***SE#*** represents different pose estimation scenarios.

		R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
SE1	PCKh[1]	13.47	86.6	74.04	64.79	74.77	83.82	85.17	60.18	88.62	87.66
	AUC	4.42	41.46	35.15	33.57	36.80	38.45	42.37	29.05	50.25	41.30
SE2	PCKh[1]	39.57	89.58	87.99	90.67	89.97	91.00	90.89	89.79	90.32	91.97
	AUC	19.62	62.36	66.55	69.05	54.98	67.86	64.71	63.57	54.52	63.61
SE3	PCKh[2]	79.93	90.89	91.76	92.95	79.09	91.74	92.72	93.64	92.74	95.55
	AUC	25.58	35.37	33.06	37.53	22.34	31.49	32.26	38.64	29.78	39.72

[1] SE1 and SE2 does matching threshold to $26\ pixels$. [2] SE3 does matching threshold to $200\ mm$.

Table 5.6: Evaluation results (PCKh@0.5) per joint group in the first three experimental scenarios. Each ***R#*** represents a sub-evaluation for the assessment of the influence of mixing real and synthetic datasets. Each ***SE#*** represents different pose estimation scenarios.

		R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
SE1[6]	Head[1]	12.81	88.76	96.77	99.63	57.55	97.72	99.04	99.71	93.30	99.73
	Shoulder[2]	26.41	92.85	96.82	99.07	85.19	97.86	99.29	99.32	96.54	99.74
	Elbow[3]	12.36	92.09	90.29	56.98	82.66	86.20	82.33	50.11	92.57	82.44
	Wrist[4]	0.42	3.38	4.94	8.60	2.83	5.22	7.66	9.61	5.04	10.52
	Hip[5]	5.57	99.79	45.81	29.21	91.95	82.37	86.86	16.36	99.88	94.00
SE2[6]	Head[1]	36.82	99.58	99.37	99.50	99.68	99.63	99.53	99.90	97.13	99.88
	Shoulder[2]	51.15	98.50	99.37	99.57	99.00	99.41	99.42	99.87	97.97	99.82
	Elbow[3]	48.91	89.05	79.94	92.98	91.39	94.63	93.96	87.38	94.83	97.00
	Wrist[4]	1.99	2.99	4.35	4.19	3.03	4.58	5.33	6.48	7.12	8.92
	Hip[5]	34.71	99.99	99.18	99.99	99.97	99.97	99.96	99.14	99.97	100.00
SE3[7]	Head[1]	89.34	98.95	99.11	99.35	83.25	98.89	99.17	99.58	98.94	99.95
	Shoulder[2]	86.77	98.72	98.95	99.12	84.98	98.18	98.65	99.22	98.98	99.72
	Elbow[3]	71.53	77.11	79.59	85.00	76.66	82.50	83.86	87.15	87.09	92.70
	Wrist[4]	66.91	71.68	73.69	74.70	70.20	74.45	76.10	76.63	72.86	81.43
	Hip[5]	81.10	99.67	99.83	100.00	78.00	98.24	99.49	99.76	99.36	99.76

[1] Head uses head and neck joints. [2] Shoulder uses rshoulder, lshoulder and chest joints. [3] Elbow uses relbow and lelbow joints. [4] Wrist uses rwrist and lwrist joints.

[5] Hip uses rhip, lhip and pelvis joints. [6] SE1 and SE2 does matching threshold to $26\ pixels$. [7] SE3 does matching threshold to $200\ mm$.

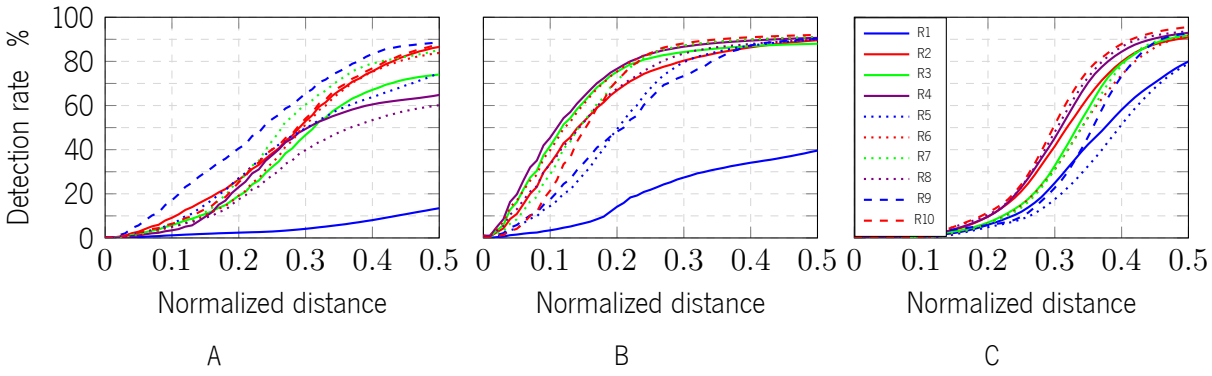


Figure 5.15: PCKh total for all sub-evaluations, $R\#$, and the three first experimental scenarios, $SE\#$: (A) 2D pose estimation from depth images (SE1); (B) 2D pose estimation from point cloud (SE2); and (C) 3D pose estimation from 2D pose (SE3). Color gradient represents synthetic data increase with constant real data. Continuous, dashed and dotted lines represent increasing amounts of real samples (for the same real-synthetic ratio).

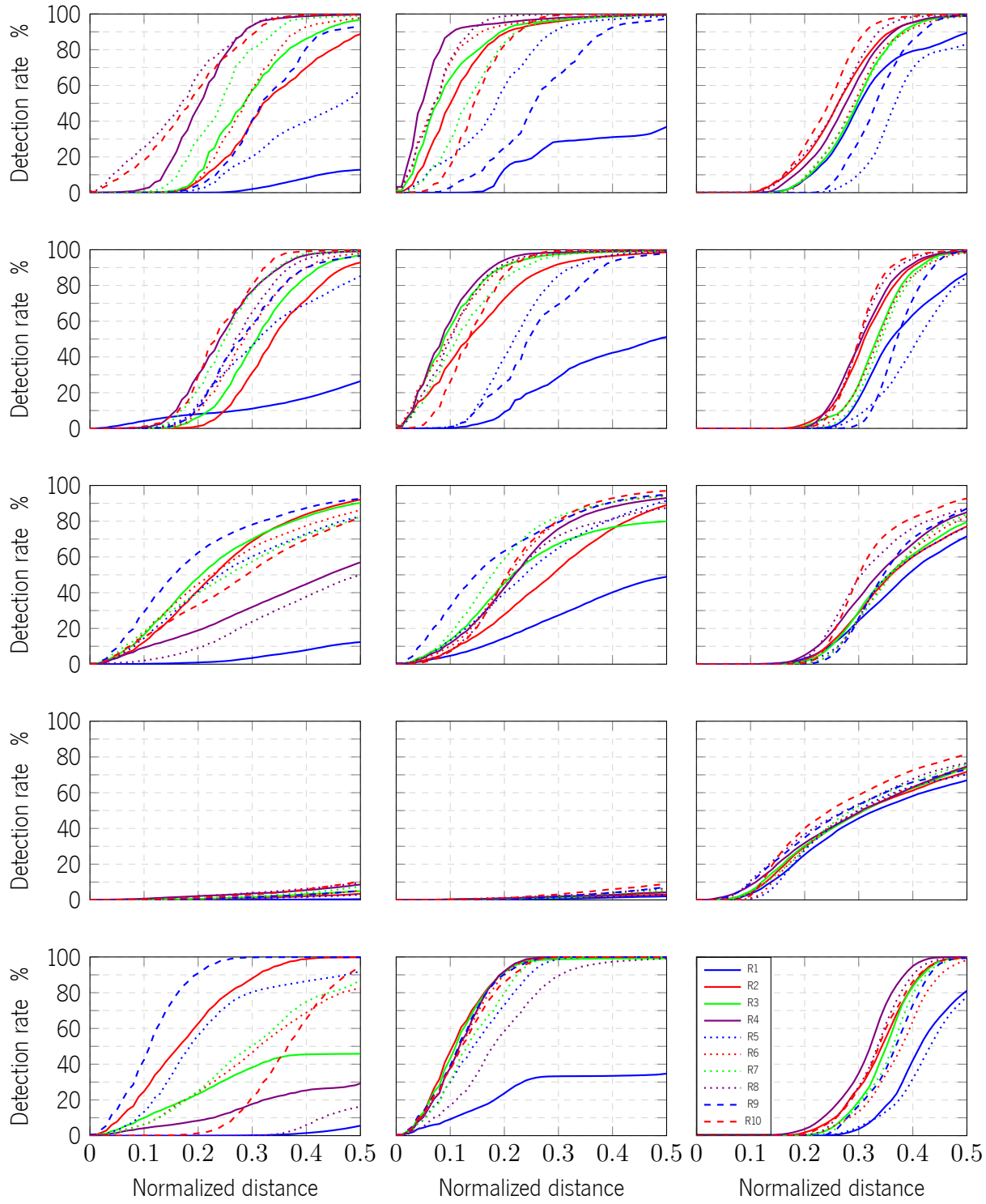


Figure 5.16: PCKh for all sub-evaluations, $R\#$, and the three first experimental scenarios, $SE\#$: (1^{st} column) 2D pose estimation from depth images (SE1); (2^{nd} column) 2D pose estimation from point cloud (SE2); (3^{rd} column) 3D pose estimation from 2D pose (SE3); (1^{st} row) Head; (2^{nd} row) Shoulder; (3^{rd} row) Elbow; (4^{th} row) Wrist; and (5^{th} row) Hip. Color gradient represents synthetic data increase with constant real data. Continuous, dashed and dotted lines represent increasing amounts of real samples (for the same real-synthetic ratio).

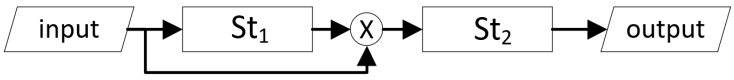


Figure 5.17: Inference pipeline for the experimental scenario SE4. Normalized point-cloud is used as input for the first U-Net stage, $\boldsymbol{St_1}$, and then masked with its output. The second U-Net stage, $\boldsymbol{St_2}$, uses the masked input and inferes the body parts' labels.

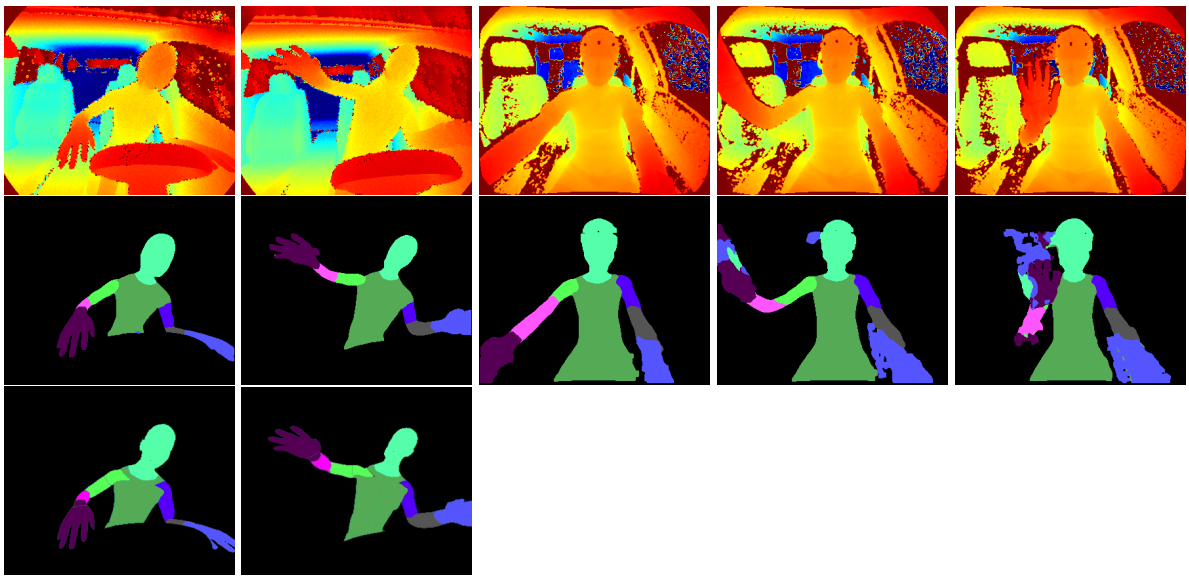


Figure 5.18: SE4 visual representation: first row represents the point cloud input features (represented in depth frame to improve understanding); middle row represents infered body parts' segmentation; and bottom row represents the label frame. The first two columns represent synthetic samples from the MoLa S25k InCar Dataset (not used in training), while the last three columns represent real samples from the MoLa R10k InCar Dataset (no label frames available).

5.4 Discussion

In this chapter, a toolchain capable of generating realistic synthetic human body pose and image data for the in-car scenario was presented. The toolchain is able to achieve a high throughput for data generation. A synthetic dataset generated by the toolchain is also made publicly available. It was proved that it is possible to automate the generation of valid synthetic data for body pose detection algorithms, focused on in-car environment. The toolchain follows a different approach than others that are focused in real [49, 51] and hybrid data [70, 74], by simplifying the procedure of generating large amounts of valid data with limited human, time and hardware resources. Although there is another purely synthetic toolchain [73], to the author's best knowledge, this is the first toolchain focused specifically in the in-car scenario.

In terms of data validation, Figures 5.15, 5.16 and Table 5.5 demonstrate the interest in including synthetic data in ML training, showing PCKh improvements in almost all experiments when adding synthetic data. Some experiments showed that higher ratios would induce overfitting to synthetic data (e.g.

$\boldsymbol{SE1{:}R2 > R3 > R4}$, $\boldsymbol{SE1{:}R7 > R8}$). However, this behaviour was not visible across all experiments, e.g. in $\boldsymbol{SE2}$ addition of synthetic samples seem to always improve accuracy, which may be related with the link between increased complexity of the input (3-channel matrix compared to the 1-channel depth image) and the need for larger number of training samples. Interestingly, besides demonstrating the interest of synthetic samples for increased algorithmic accuracy, the present results also seem to suggest that the use of a 3D point cloud as input may lead to a better pose inference when compared to networks using depth images (see for example the higher AUC values for $\boldsymbol{SE2}$ compared with $\boldsymbol{SE1}$, meaning more detections for a lower normalized distance). A similar improvement in inference performance when adding synthetic samples was also observed in $\boldsymbol{SE3}$. In the end, for the in-car 2D body pose estimation problem, we achieved higher PCKh@0.5 total score in $\boldsymbol{SE2{:}R10}$ (i.e. full dataset training) of 91.97% and corresponding AUC of 63.61%. For the in-car 3D body pose estimation problem, considering both metrics (PCKh@0.5 total and AUC), we achieved higher score in $\boldsymbol{SE3{:}R10}$ with 95.55% and AUC of 39.72%, with an increase of 3% and 10% in PCKh@0.5 and AUC, respectively, when compared to $\boldsymbol{SE3{:}R9}$ (i.e. real data only), meaning a better 3D joint inference across all thresholds. Finally, Table 5.6 summarizes the results for individual joints, being possible to conclude that joints frequently present at the image's limits (wrists and hips) are the most problematic. This may be related with the lower number of training samples with these joints visible (as they are more frequently outside of the camera's FoV or in the camera's deadzone). Overall, these results prove the validity of the data being generated and its interest for human body pose detection algorithmic development.

In terms of body parts' segmentation, $\boldsymbol{SE4}$ experimental scenario showed that pure synthetic training is still capable of achieving good results in real data, however requiring increased method complexity to cope with lack of noise realism in synthetic data. Indeed, inferred results for unseen synthetic samples are quite similar to the known label frame (first two columns of Figure 5.18), as a result of similar noise and background features, while results for real samples present in some cases sub-optimal performance due to distinct noise/background image features (look to the inference error in hands and head for the last 3 columns in Figure 5.18). Notwithstanding, from the results obtained for $\boldsymbol{SE1}$ to $\boldsymbol{SE3}$ (mix of real with synthetic samples), it is expectable a noticeable improvement if a few real samples are included during training (which was not available in the present experiments due to lack of ground-truth labels).

5.5 Conclusions and future work

In this work, a novel toolchain for the generation of realistic synthetic images for human body pose detection in an in-car environment is presented. A synthetic dataset generated by the toolchain is also made publicly available. The toolchain demonstrated to be computationally efficient (up to 4 generated frames per second), while demonstrating its potential for increased algorithmic accuracy during body pose estimation in an in-car scenario.

In terms of dataset generation performance, several considerations can be made for improvements in future work. Specific Python functions (Algorithm 5.2.9, equation 5.5) can be rewritten to take advantage of specific hardware resources or better performing programming languages. In terms of automation, extra customizations can be added into the toolchain, namely human body and camera Gaussian translation and rotation. This would allow for faster data customization with reduced manual interaction. Another important aspect is scene realism, as discriminative algorithms seem to improve their accuracy proportionally to the training data realism. In this regard, future work may focus on improving the ToF noise characterization or the used NST methods, as well as the RGB image rendering. The ability to synthetically recreate human behaviour would be another important feature, enlarging the applicability of our dataset towards other monitoring tasks (like action recognition). Hereto, fusion of real human motion capture data with synthetic scenarios could be employed. However, issues such as collision detection between animated models and synthetic car models would have to be handled. Besides the currently supported pose and segmentation maps, another relevant output to be added would be the gaze for each human model.

Chapter 6

Algorithm Development

As it was shown in Chapter 1, this chapter is sequential to chapters 4 and 5 in terms of development. However, its initial tasks are also done in parallel to others. Preliminary versions of the work presented in this chapter were presented in paper [82].

In the first section of this chapter, the selection and evaluation of multiple state-of-the-art human body pose detection methods is shown. In short, four methods were selected (two machine learning and two deep learning based) and evaluated. Evaluation was performed using a publicly available dataset.

In the second section of this chapter, the best algorithm from Section 6.1 is customized and evaluated with a manually labeled dataset. For customization, several changes were made: (1) removal of refinement stages for computational optimization; (2) addition of third branch for label detection; (3) training 3D data augmentation sub-algorithm; and (4) training hiper-parameters tuning. The modified algorithm is then evaluated, followed by a discussion of its results and main conclusions taken.

Contents

6.1 Algorithm Selection

To develop the human body pose detection method, a strategy was defined. First, it was considered that the body pose detection algorithm must have a discriminative basis owing to its advantages over the generative approaches. Secondly, it was decided not to focus only in one methodology, avoiding narrowing the scope of the investigation. In this sense, the two types of discriminative approaches, namely traditional machine learning and deep learning, are being studied. Third, they were selected considering their computational performance, accuracy and feature/label resemblance to our scenario. The research strategy can be summarized by the following blocks:

- Evaluate algorithms with traditional machine learning:
  - Evaluate the Deformable Part Models (DPM) [21];
  - Evaluate the Random Tree Walks (RTW) [23];
- Evaluate algorithms based on deep learning:
  - Evaluate the You Only Look Once (YOLO) [24];
  - Evaluate the Part Affinity Fields (PAF) [25].

6.1.1 Selected Methods

As already mentioned, the strategy to develop the body pose detection algorithm, involves implementing both traditional machine learning and deep learning methods to evaluate the best approach for the present thesis. Within the traditional machine learning, the DPM [21] and RTW [23] methods were implemented. Regardind Deep Learning (DL), the YOLO [24] and PAF [25] methods were implemented. In this section, the fundamental concepts of these methods are presented.

6.1.1.1 Deformable Part Models (DPM)

The DPM method is an object detection system that uses local appearances and spatial relations to recognize generic objects in an image. Generically, this method consists in the definition of a model that represents the object, being this model constructed through the definition of a root filter (for the entire object) and through a set of part filters (for the different parts of the object). These filters are used to study the features of the image. In specific, the Histogram of Oriented Gradients (HoG) features are computed inside

each filter to represent an object category. The HoG descriptor computes the gradients of a region of the image, assuming that the object within the image can be described by its intensity transitions. In the DPM method, a sliding window approach is used, being the filter applied at all positions of the image. In order to create the final model, a discriminative approach is used, where the model is learned from labelled data using only bounding boxes around the object and its parts to represent the filters. This discriminative part of the method is performed normally using Support Vector Machine (SVM). After this training phase, the model is used to detect the object in the test image. The detection is performed by convolving the trained model with the feature map of the test image and by selecting the image with the highest convolution score, as seen in Figure 6.1. Note that, despite the discriminative basis of the DPM method, this test phase can be interpreted as fitting the model in the image, which involves generative concepts. In this sense, it is possible to consider the DPM method as a hybrid methodology.

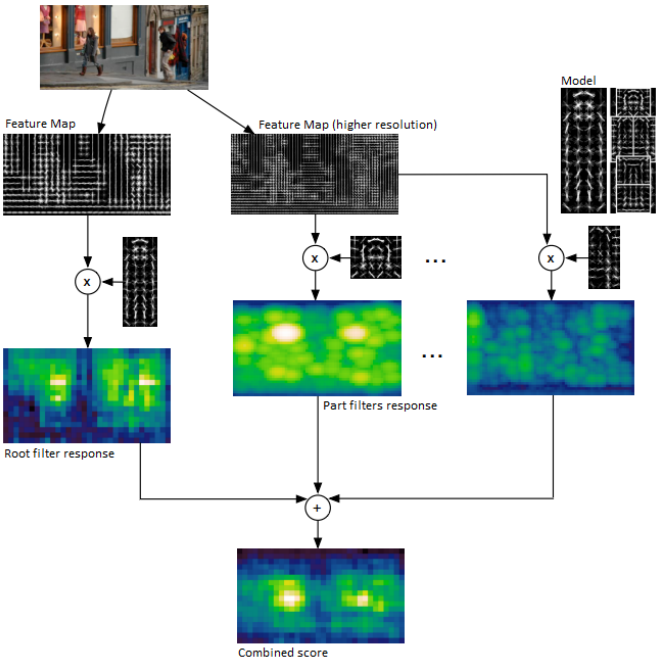


Figure 6.1: Felzenszwalb et al. matching process in the testing phase. Image adapted from [21].

To deal with the variety of the object (e.g. the different poses in a human), a mixture of models was proposed. In this approach, instead of using only one model to represent the object, a set of models are constructed by applying different combinations between the deformable parts filters. This concept was used in [111] for human body pose detection. In this method, the DPM concepts were used to represent the human body. Moreover, it also uses a pictorial structure framework to represent the model, allowing to represent the human body by a collection of parts arranged in a deformable configuration. In this pictorial structure framework, the correct relationships between the different human body parts are established by applying geometrical constraints that model the human body.

The DPM method has been used mostly for RGB images (Figure 6.2).

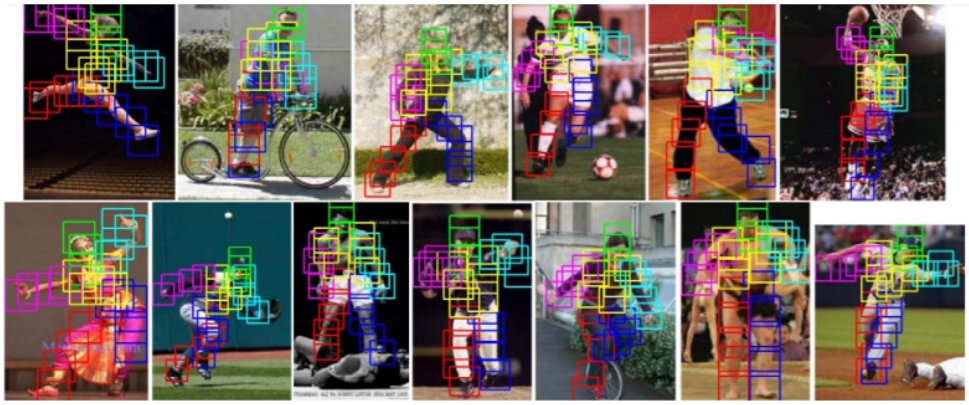


Figure 6.2: Felzenszwalb et al. pose detection results in RGB images using DPM method. Image adapted from [21].

6.1.1.2 Random Tree Walks (RTW)

The RTW implementation estimates 3D joint positions from depth images. This work is an extension of a previous one, proposed in [22]. The method proposed by Shotton et al. is based in an efficient Decision Tree (DT) algorithm for pixel-wise classification/regression. Using simple discriminative depth comparison image features, this method achieves high-computational efficiency (real time - 200 frames per second (fps)) with 3D translation invariance. Due to the clear advantages of this method in relation with the other state-of-the-art methods (i.e. trade-off between accuracy and computation time), this strategy was the baseline for a set of other implementations, namely RTW. Next, the Shotton framework is introduced, which will support the explanation of the RTW algorithm later. As already mentioned, Shotton's algorithm is based in a DT strategy. A detailed explanation about decisions tree can be found in [112]. Briefly, in the training stage of a DT, all the data (i.e. depth images) are introduced in the root of the tree (Figure 6.3).

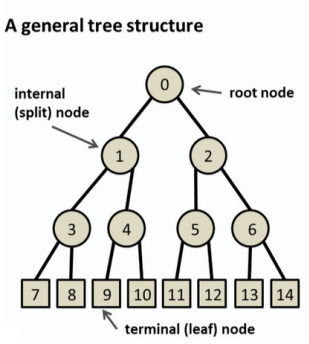


Figure 6.3: A tree is a set of nodes and edges organized in a hierarchical fashion. A DT is a tree where each internal node stores a weak (or split) function to be applied to the incoming data. Each leaf stores the final answer (predictor).

Next, in each node, a weak function represented by specific parameters is used to separate the data in two groups as different as possible from one another. In the Shotton's method, these parameters allow the computation of a feature given by the difference between the depth intensity in two relative positions (i.e. given by the two defined offsets) from each pixel, $\boldsymbol{u}$, (Figure 6.4).

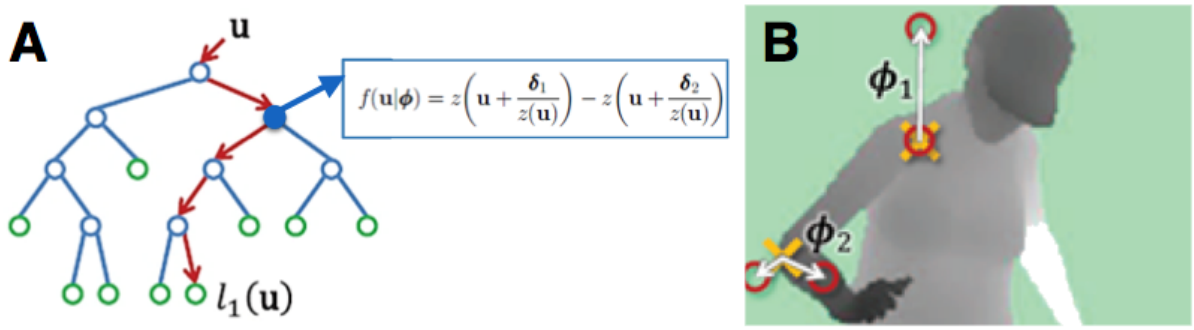


Figure 6.4: Shotton's DT example. (A) In each node, a feature given by the difference between the depth intensity of two relative positions (given by the offsets $\boldsymbol{\delta_1}$ and $\boldsymbol{\delta_2}$) are computed. The value is then compared with a threshold, following to the left or right side of the tree; (B) Two feature examples. The circle denotes the relative points to the evaluated pixel, here represented by a yellow cross. Image adapted from [22].

After that, the result is compared with the threshold, in order to define if it will follow the right or left side. This strategy is used in the following nodes until reaching a leaf. When the data reaches a leaf, the information saved in each is different according with the problem type: 1) in the case of a classification problem, the output of the leaf is a set of probabilities that represent each labelled body part; 2) in a regression problem, the output of the leaf is a relative distance to each joint of interest (Figure 6.5). Thus, in the end of the training phase, the information stored is the set of parameters that define each node and the information that represent each leaf.

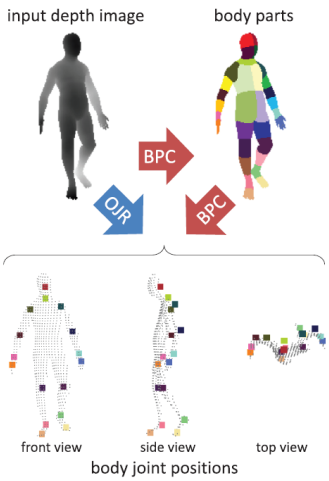
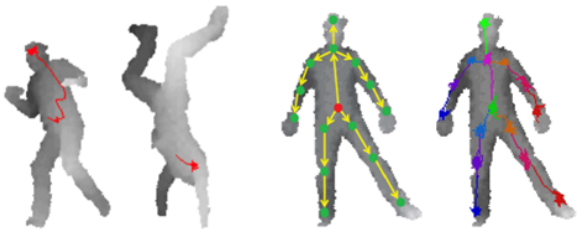


Figure 6.5: Shotton's DT implementations: OJR is a regression tree that stores in each leaf an offset for each joint; BPC is a classification tree that stores, in each leaf, the probability of a pixel being a specific body part. Note that, the estimation of 3D joint positions using BPC is not direct (needs an additional algorithm). Image adapted from [22].

In the testing phase, it is expected to evaluate separately each image pixel, $\boldsymbol{u}$. In each node of the tree, the feature given by the set of parameters learned in the training phase. After that, the feature is compared with the threshold in order to define if it will follow the right or left side. This process continues until it reaches the leaf. In the leaf, the information that is stored, will permit a classification/regression of each pixel, $\boldsymbol{u}$. In conclusion, the method uses simple depth pixel comparison features (three arithmetic operations in each node) making it possible to run in real time.

As mentioned, RTW is an extension of Shotton's method, differing in the fact that: 1) it does not apply a pixel-wise regression in all pixels from an image; 2) it trains a tree to estimate the direction that a random point has for a specific joint, instead of a distance/probability [23]. Briefly, the RTW uses the same tree structure as Shotton, with simple depth pixel comparison features in each node. Thus, in the training-phase, it will learn the same parameters (i.e. 2 offsets and 1 threshold). The main difference is that it learns (i.e. stores in the leaf) the direction that random points have for each specific joint, instead of a distance as Shotton. In the testing phase, as in Shotton's implementation, the features given by the difference between the depth intensity in two relative positions from $\boldsymbol{u}$ is computed. However, note that in RTW just one pixel is evaluated in each iteration. When it reaches a leaf, it assigns the direction that this specific point has to a specific joint in the human body. Next, using this direction, the RTW method will iteratively take steps towards the joint (Figure 6.6A). This method runs in a hierarchical manner, which means that the result position for a joint will be used as the initial point for the next joint position to be computed (Figure 6.6B). Concluding, RTW classifies much less points than Shotton's strategy, achieving an improved computation time (i.e. 1000 fps). Moreover, this large computational gain is achieved without decreasing accuracy in 3D joints estimation. Due

to its clear advantages, RTW implementation has the potential to reach the aims of this thesis.



A B

Figure 6.6: RTW method: (A) The red lines represent the RTW method trained to find the head position, (B) the method starts estimating the belly 3D joint position. The other ones are estimating hierarchically as represented in the figure. Image adapted from [23].

6.1.1.3 You Only Look Once (Yolo)

YOLO is a deep learning method for detecting objects. Basically it reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities (Figure 6.7) [24].

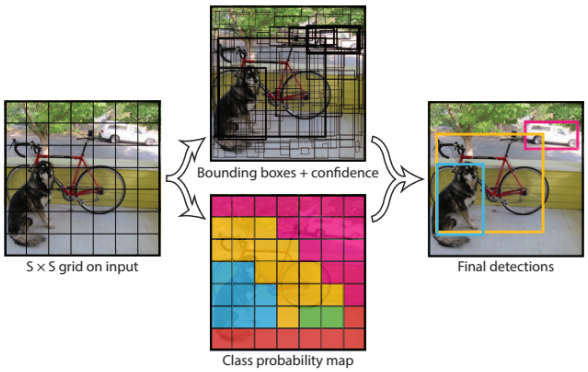


Figure 6.7: YOLO method. This detection method works as a regression model. It divides an image into a $\boldsymbol{SxS}$ grid and for each grid cell predicts $\boldsymbol{nB}$ bounding boxes, confidence for these boxes, and $\boldsymbol{nC}$ class probabilities. Image adapted from [24].

YOLO is a single convolutional network that simultaneously predicts multiple bounding boxes and class probabilities for those boxes. It trains on full images and directly optimizes detection performance. This type of method has several advantages against others methods, including:

- YOLO frames detection as a regression problem, running a new image in its neural network to predict detections. The base network runs at 45 fps and a fast version (i.e. lighter) runs at more than 150 fps. This way, YOLO can run in real time video.

- Since it sees the entire image during training and testing, it implicitly encodes contextual information about the classes and therefore considers the image globally when making predictions.

- It learns generalized representations of objects. The moment is trained and tested in natural images, the YOLO can overcome the best detection methods, such as DPM and R-CNN. Since YOLO is quite generalist, it is less likely to break when applied to new fields, such as depth images.

This neural network has all the detection components uniformed, and uses all the features of the image to predict a bounding box. It also predicts all the bounding boxes for all classes in a single image simultaneously. This method divides the input image in a S x S grid. If the center of the object is inside a grid cell, this cell is responsible for detecting this object. Each grid cell predicts $\boldsymbol{nB}$ bounding boxes and the confidence score for each one of them. The box confidence score $\boldsymbol{\hat{C}_c}$, of the box $\boldsymbol{bb}$ in cell $\boldsymbol{c}$, is formally defined by equation 6.1. If no object exists in the cell, then the confidence score should be zero. Otherwise, the confidence score is equal to the Intersection Over Union (IOU) between the predicted bounding box and the ground-truth. Each bounding box predicts 5 values: $\boldsymbol{t_x}$, $\boldsymbol{t_y}$, $\boldsymbol{t_w}$, $\boldsymbol{t_{he}}$ and confidence. The $\boldsymbol{t_x}$ and $\boldsymbol{t_y}$ are coordinates of the center of the bounding box relative to the size of the grid. The width $\boldsymbol{t_w}$ and height $\boldsymbol{t_{he}}$ are predicted relative to the entire image. Finally, the confidence score represents the IOU between the predicted bounding box and the ground-truth.

$$\hat{C}_c = P_r(object) \cdot IOU\frac{truth}{pred} \tag{6.1}$$

Each cell in the grid predicts $\boldsymbol{nB}$ probabilities of conditional class, $\boldsymbol{P_r(class_c \mid object)}$. These probabilities are conditioned to the cell containing an object. Only one set of class probabilities per cell is predicted, regardless of the number of bounding boxes $\boldsymbol{nB}$. When the image is being tested, the conditional class probabilities and the confidence predictions are multiplied, which allows to assign the specific confidence score for each bounding box (equation 6.2). Thus, measuring the confidence of classification and localization.

$$\begin{gathered} P_r(class_c \mid object) \cdot P_r(object) \cdot IOU\frac{truth}{pred} = P_r(class_c) \cdot IOU\frac{truth}{pred} \\ conditionalclassprobability \cdot boxconfidencescore \equiv classconfidencescore \end{gathered} \tag{6.2}$$

6.1.1.4 Part Affinity Fields (PAF)

PAF method is a deep learning-based framework for multi-person pose estimation in 2D images (Figure 6.8) [25]. This method uses a two branches network configuration for jointly detecting the joints' positions

and the associations between them (Figure 18). In the first branch of the method, a feed-forward network predicts the confidence maps of body parts' locations which corresponds the probability maps (Figure 6.8B). These probability maps are a representation of the confidence of a joint position occurring at each pixel location, expressed as a gaussian function. In the second branch of the network, the part affinity vector fields are constructed, encoding the degree of association between the parts (Figure 6.8).

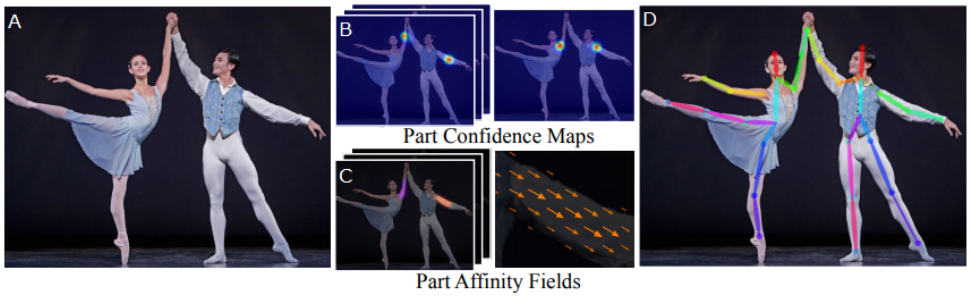


Figure 6.8: PAF method: (A) original image, (B) part confidence map, (C) part affinity fields, and (D) body pose detection result. Image adapted from [25].

The part affinity fields allow to assemble the joint positions to form a full-body pose, being constructed for each body limb and encoding both location and orientation information. The two branches of the network are refined along several stages in an iterative process. Moreover, the predictions of both branches in each stage of the method are used in the predictions of the following stage, allowing to boost the refinement process (Figure 6.9).

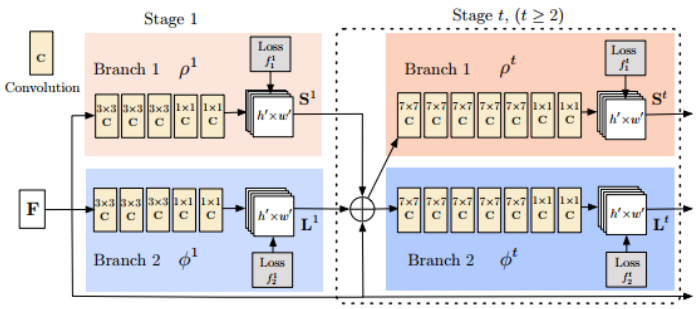


Figure 6.9: PAF method. Architecture of the two-branch deep-learning network. Image adapted from [25].

Among the major advantages of this method are its real time performance and high accuracy to detect multiple persons. In fact, the PAF branch was designed to better deal with an image with more than one person. In this regard, it is not needed the implementation of a prior person detector method to detect the joint positions for each person afterwards. This allows to avoid bad detections of the person detector method while decreasing the computation time. As a disadvantage, this method requires a great deal of data for the training process. Moreover, the PAF method has only been used for RGB images. However, due to its clear

advantages, it has the potential to reach the aims of our work, and therefore, the investigation team decided to exploit and implement this method for depth images, being the results presented later in this report.

6.1.2 Evaluation

In this section, the implemented steps related with the human body pose detection algorithm are described. Both DPM, RTW, YOLO and PAF were implemented and tested, being the results presented later in this section.

6.1.2.1 Dataset

Due to the fact that the work in chapters 4 and 5 was still ongoing and no datasets were available, it was needed to search for a public dataset that could be used to evaluate and compare the implemented methods. Between the different datasets found in the literature, it was decided to use the Towards Viewpoint Invariant 3D Human Pose Estimation (ITOP) dataset [62], due to its similarities in features and ground-truth. The ITOP is a publicly available depth-based human pose dataset that has been used by some state-of-the-art human pose methods. The dataset consists of 20 people performing 15 action sequences each. Each depth map (Figure 6.10A), and its respective point cloud (Figure 6.10B), is labelled with the ground-truth that consists in the pixel-world 2D and real-world 3D joint locations (Figure 6.10C). Moreover, the dataset is divided in training and test sets, containing each set 40k and 10k images respectively.

A B C

Figure 6.10: ITOP dataset: (A) depth image, (B) point cloud, and (C) ground-truth joints.

One drawback found in the ITOP dataset is that some ground-truth joint positions do not fit correctly with the human body, which can be explained by errors during the ground-truth generation. In this sense, a method to perform the correction of the displaced joints was first implemented (Algorithm 6.1.1). The goal of the method is to move the joints placed in the background of the image to the interior of the human body. For that, the initial step is to compute a mask of the human silhouette (Figure 6.11).

Algorithm 6.1.1 ITOP DATASET CORRECTION

initialize:

Get the depth and body pose samples from ITOP dataset

$depth_{t',x,y} \leftarrow ITOP$

$(Pxy_J^C)_{t'} \leftarrow ITOP$

Get the foreground, and remove the background $\boldsymbol{RG_{t',x,y}}$. Use the regiongrowing method using the head position $\boldsymbol{(Pxy_{head}^C)_{t'}}$ as seed position

$RG_{t',x,y} \leftarrow regiongrowing(depth_{t',x,y}, seed \leftarrow (Pxy_{head}^C)_{t'})$ [113] (**Figure 6.11B**)

Remove the floor

$$RG_{t',x,y} \leftarrow \begin{cases} 1, \\ 0, y \geq 220 \end{cases}$$ (**Figure 6.11C**)

Get the pixel connectivity $\boldsymbol{CC_{t'}}$

$CC_{t',x,y} \leftarrow bwconncomp(RG_{t',x,y})$ [114]

Preserve the human silhuete by crossing the pixel connectivity with the head pixel position

$RG_{t',x,y} \leftarrow RG_{t',x,y} \cap CC_{t',x,y} \cap (Pxy_{head}^C)_{t'}$ (**Figure 6.11D**)

Get the new joint position considering the minimum distance of the joint actual position and the mask

for each t' in $(Pxy_J^C)_{t'}$ **do**

  for each j in $(Pxy_J^C)_{t'}$ **do**

    for each x, y in $(RG_{t',x,y} \cap 1)$ **do**

      $(Pxy_J^C)_{t'} \leftarrow \underset{x,y}{argmin}(d((Pxy_J^C)_{t'}, \{x, y\}))$ (**Figure 6.12B**)

    end for

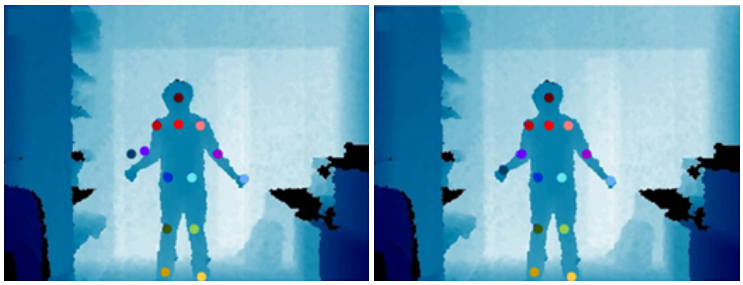
  end for

end for

A B C D

Figure 6.11: Human silhouette mask construction: (A) original image, (B) region growing result, (C) floor elimination, and (D) final mask.

This mask is generated by a region growing approach [113], using the torso position as initial seed point. After performing the region growing method, a floor elimination strategy is applied to remove the floor regions that are wrongly classified as human silhouette by the region growing method. Once the human body mask is generated, the joints that are placed outside the human silhouette are replaced in the nearest position of the mask. In Figure 6.12, it is possible to visualize the ground-truth of one example image before and after the joint position correction.



A B

Figure 6.12: Correction of ITOP dataset: (A) joint positions before joints correction, and (B) joint positions after joints correction.

6.1.2.2 Metrics

The evaluation metrics used to assess the methods accuracy were:

- mean average distance, $\boldsymbol{mAD}$, (equation 6.3): mean average of the Euclidean distance between the joint position given by the method ($\boldsymbol{P_{ALG} \equiv (Pxyz_J^C)_{t'}}$), and the joint position in the ground-truth ($\boldsymbol{P_{GT} \equiv (Pxyz_J^C)_{t'}}$).

$$\begin{aligned} d_j &= d(P_{ALG}, P_{GT}) \\ \boldsymbol{mAD} &= \frac{\sum_{j=1}^{J} \frac{\sum_{t'=1}^{T} d_j}{T}}{J} \end{aligned} \tag{6.3}$$

- mean average distance for valid joints, $\boldsymbol{mAD^{10cm}}$, (equation 6.4): mean average distance after elimination of wrong detections, $\boldsymbol{TP}$. Defines if the joint was correctly detected. If the Euclidian distance, $\boldsymbol{d_j}$, between the joint position given by the method, $\boldsymbol{P_{ALG}}$, and the joint position of the ground-truth, $\boldsymbol{P_{GT}}$, is less than 10 cm, the joint is considered correctly detected. If this condition is not verified the joint is considered wrongly detected.

$$\begin{aligned} label &= \begin{cases} TP, P_{ALG} \neq \emptyset \wedge P_{GT} \neq \emptyset \wedge d_j \leq 10 \\ FN, P_{GT} \neq \emptyset \wedge ((P_{ALG} \neq \emptyset \wedge d_j > 10) \vee (P_{ALG} = \emptyset)) \\ FP, P_{ALG} \neq \emptyset \wedge P_{GT} = \emptyset \\ TN, P_{ALG} = \emptyset \wedge P_{GT} = \emptyset \end{cases} \\ \boldsymbol{mAD^{10cm}} &= \frac{\sum_{j=1}^{J} \frac{\sum_{t'=1}^{T} d_j(TP))}{T}}{J} \end{aligned} \tag{6.4}$$

- mean average precision, $\boldsymbol{mAP^{10cm}}$, (equation 6.5): mean average precision of joints correctly detected by the method.

$$\boldsymbol{mAP^{10cm}} = \frac{\sum_{j=1}^{J} \frac{|d_j(TP)|}{|d_j(TP)|+|d_j(FP)|}}{J} \tag{6.5}$$

- mean average recall for valid joints, $\boldsymbol{mAR}$, (equation 6.6): mean average percentage of joints detected by the method.

$$\boldsymbol{mAR} = \frac{\sum_{j=1}^{J} \frac{|d_j(TP)|}{|d_j(TP)|+|d_j(FN)|}}{J} \tag{6.6}$$

6.1.2.3 Results

In this subsection, the result of the conducted experiments are presented. Table 6.1 presents and compares the results of each body pose detection method.

Table 6.1: Algorithmic selection evaluation.

	DPM		**RTW**	**YOLO**	**PAF**	
	RGB	**DEPTH**	**DEPTH**	**DEPTH**	**RGB**	**DEPTH**
mAD (cm)	15.61	10.69	13.22	15.70	13.34	5.37
mADv^{10cm} (cm)	5.86	4.97	4.40	5.68	4.96	3.91
mAP^{10cm} (%)	55.10	70.28	82.69	52.07	68.21	89.89
mAR (%)	100.00	100.00	100.00	96.97	100.00	100.00

Analysing Table 6.1, it is possible to understand the performance of each method for human body pose detection.

Performance for the DPM method increased when training in depth images in comparison with training in RGB. However, the RTW method outperformed the DPM method in both experiments. This may be explained by the fact that the DPM method is an hybrid approach (mix of generative plus discriminative concepts), while the RTW method is purely discriminative. This allows the RTW method to be more efficient and less dependent of a good initial model, while decreasing its susceptibility to local minima and error accumulation. In this sense, the results presented support the conclusion about the higher performance of the machine learning methods in relation with the hybrid ones. Despite the good performance of this traditional machine learning method, a higher performance was found for the deep learning methods, given that the latter do not require the extraction of hand-crafted features. The features used by the classification process are extracted

internally by the deep learning method, allowing to decrease the dependence on the user/developer expertise, and therefore, allowing to achieve more accurate results.

Regarding YOLO method, it is possible to verify that it was not able to achieve results as good as RTW. However, as mentioned in subsection 6.1.1.3, YOLO is a very generalized method, with a multitude of hyper-parameters, and may thus require a better tuning to achieve better results for depth images.

Concerning the PAF method, it is possible to verify its superior performance in comparison with the other studied methods. In fact, the PAF method achieved an average distance lower than $5.5\ cm$ and an average precision of approximately 90%, which is competitive with the results reported in state-of-the-art methods for body pose estimation in depth images. The better results can be explained by the high accuracy of its two-branch strategy with iterative refinement, which allows to refine the detection in each iteration. To better understand the performance of the most accurate detection, an extended analysis of the PAF performance is presented (Figure 6.16), when comparing with the other methods (Figures 6.13, 6.14 and 6.15). Figure 6.16 presents the PAF depth model boxplots for the metrics of $\boldsymbol{mAD}$ (Figure 6.16D), $\boldsymbol{mAD^{5cm}}$ (Figure 6.16E) and $\boldsymbol{mAD^{10cm}}$ (Figure 6.16F), allowing to analyse the performance of the method in terms of percentiles and outlier points.

Analysing Figure 6.16, it is possible to verify the good performance of the PAF depth model for all joints, similarly to the DPM and RTW depth models (Figures 6.13 and 6.14). The head joint presented the best results, while the feet joints presented the worst. This happens because the head is usually the body part with more distinguishable features in the depth image. In opposition, the intensity of the feet can be very close to the intensity of the floor where the person stands, making the method less accurate for these joints. However, it is important to notice that in this thesis the upper body accuracy has a higher significance than the lower body accuracy, since the method will be applied in depth images in a car environment.

Due to the good results achieved with the PAF method in the ITOP dataset, it was decided to use this method as the final one.

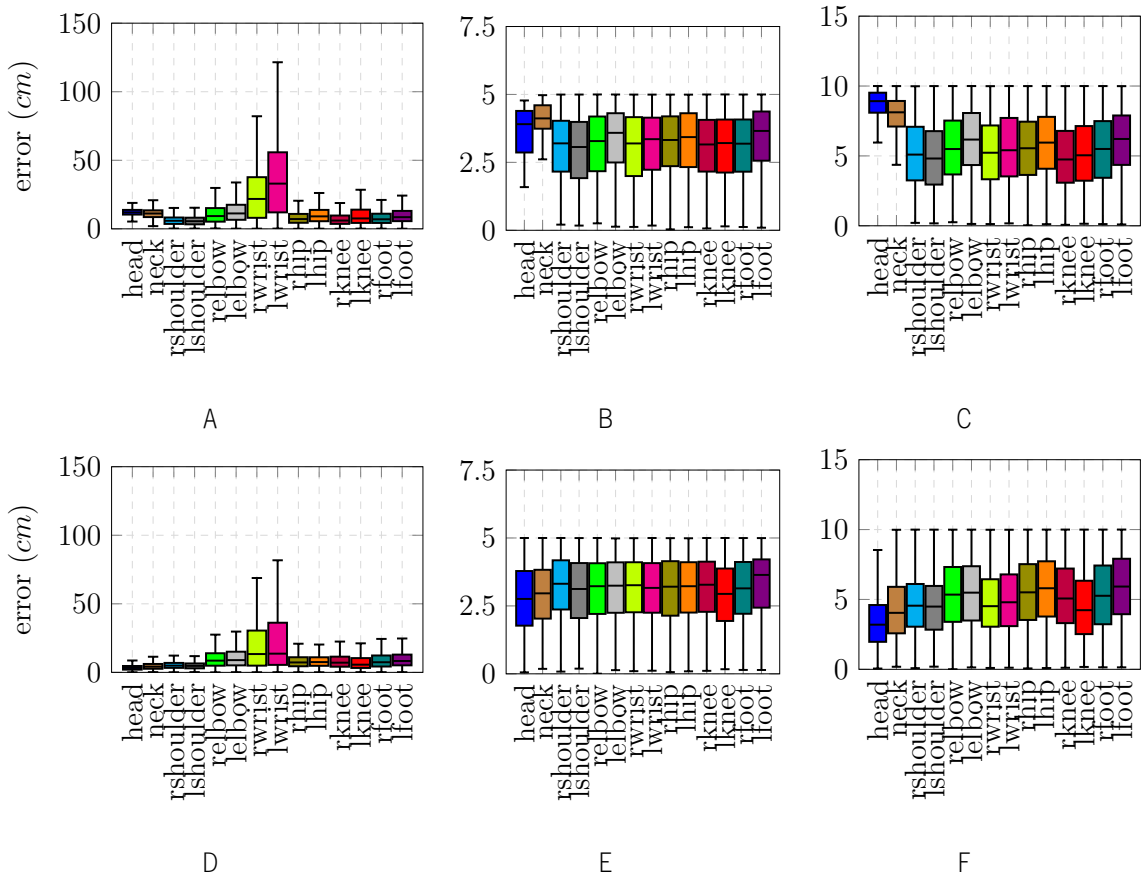


Figure 6.13: DPM error evaluation with ITOP dataset. (A - C) being the DPM with RGB model, (D - F) being the DPM with depth model, (A and D) represents the entire dataset, (B and E) represents detections under a threshold of 5 cm, and (C and F) represents detections under a threshold of 10 cm.

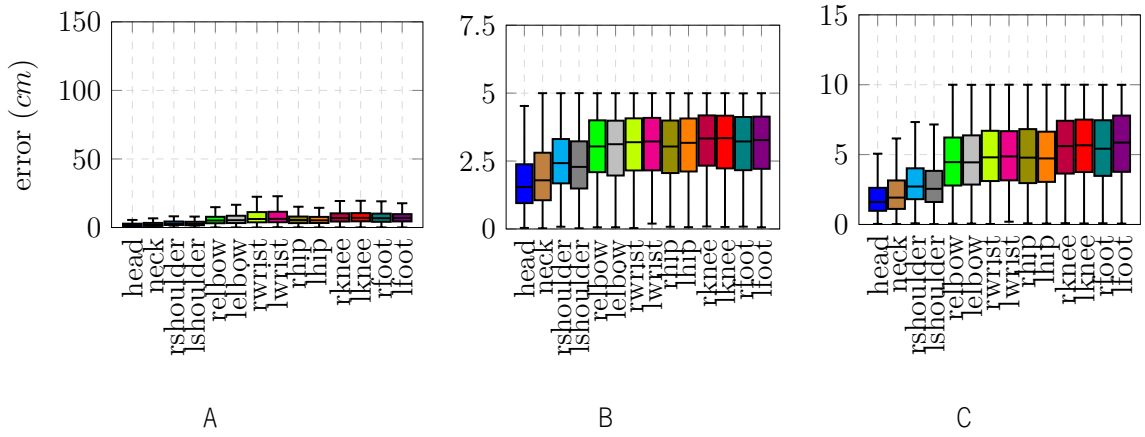


Figure 6.14: RTW error evaluation with ITOP dataset. (A - C) being the RTW with depth model, (A) represents the entire dataset, (B) represents detections under a threshold of 5cm, and (C) represents detections under a threshold of 10 cm.

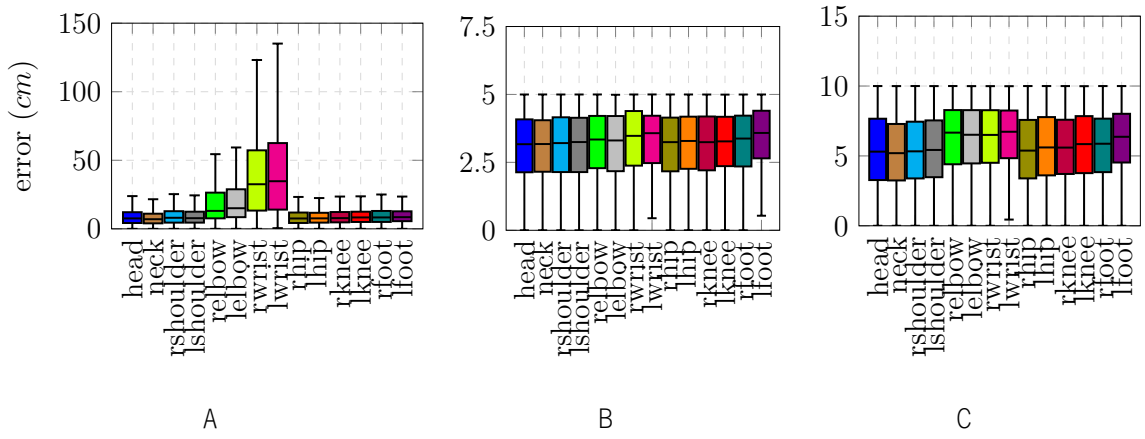


Figure 6.15: YOLO error evaluation with ITOP dataset. (A - C) being the YOLO with depth model, (A) represents the entire dataset, (B) represents detections under a threshold of 5 cm, and (C) represents detections under a threshold of 10 cm.

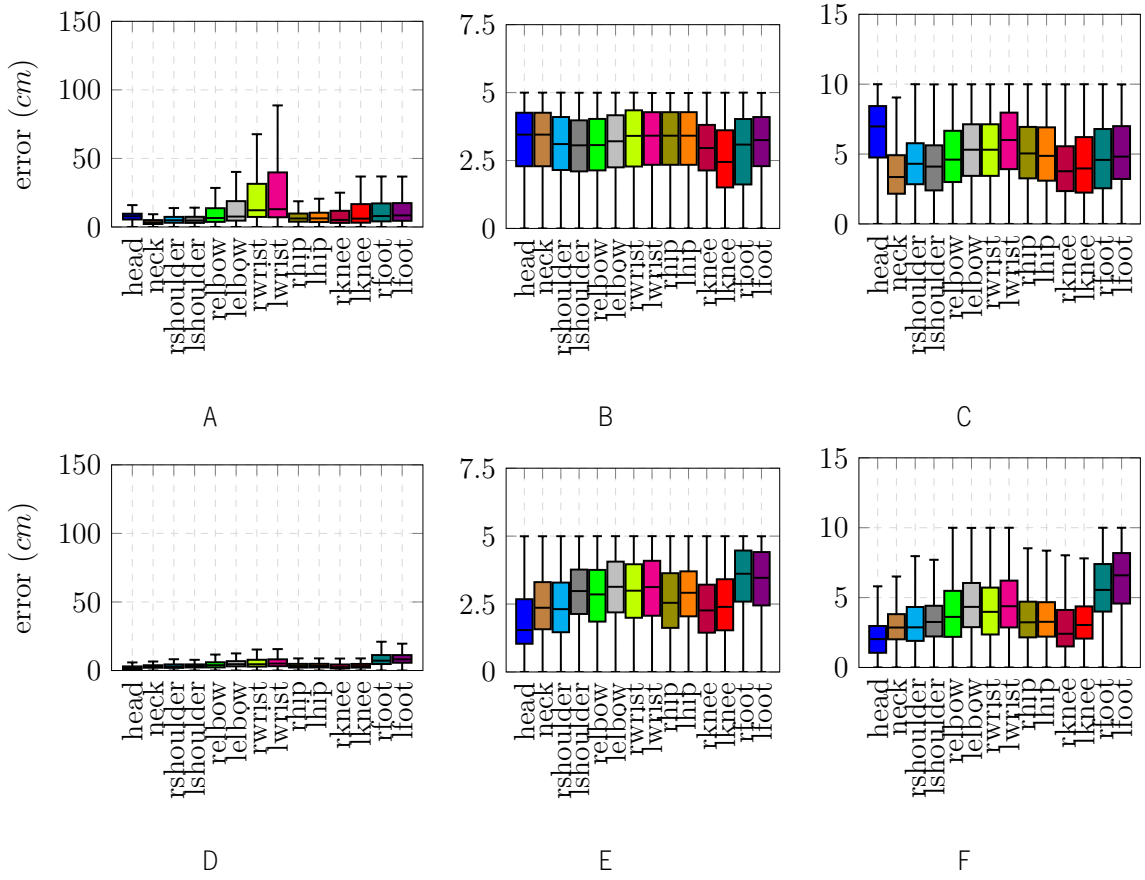
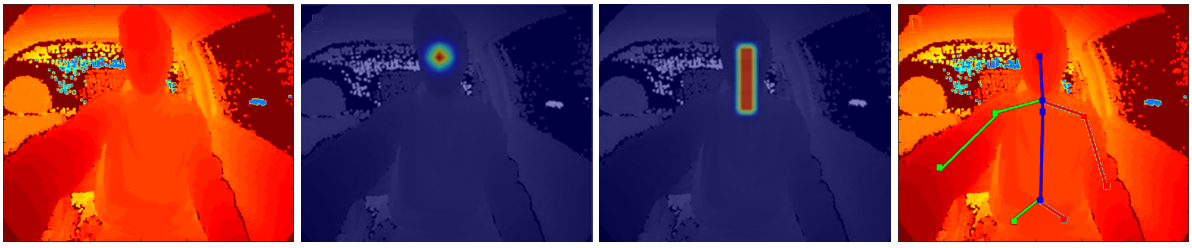


Figure 6.16: PAF error evaluation with ITOP dataset. (A - C) being the PAF with RGB model, (D - F) being the PAF with depth model, (A and D) represents the entire dataset, (B and E) represents detections under a threshold of 5 cm, and (C and F) represents detections under a threshold of 10 cm.

6.2 Algorithm Customization

6.2.1 Methods

The main goal of the proposed method was to accurately detect the human body pose of passengers inside a car in depth images. Figure 6.17 presents an overview of the proposed method, which is based on the deep learning-based body pose estimation method presented in [25] and implemented in Caffe framework [115]. The proposed method uses as input a depth image, $\boldsymbol{depth_{x,y}}$, of the driver acquired from a Time-of-Flight (ToF) camera (Figure 6.17A) and as output the location of each joint of the different human body parts, $\boldsymbol{Pxy_j^C}$, (Figure 6.17D). To obtain the joints' positions, a Convolutional Neural Network (CNN) is used to simultaneously predict a set of heatmaps (one for each body part joint, Figure 6.17B) and a set of part affinity field vectors that represent the association between the different parts (Figure 6.17C). Since the in-car environment produces occlusions of some body parts and the Field-of-View (FoV) of commercial depth sensors may not be enough to visualize all joints once the driver stands near the camera, some joints may not be detectable in the images. Thus, the proposed network also predicts if a joint is present in the image or not (henceforward called as label detection branch), which may boost the method's robustness. Considering the new Point-of-View (PoV) of the sensor and its focus on a single person, the refinement stages ($\boldsymbol{t \geq 2}$) from the original method were removed (Figure 6.9), decreasing the method's computational burden and allowing to achieve a real time performance [116].



A B C D

Figure 6.17: Overview of the proposed method. (A) input depth image, (B) heat map (output of first branch) for the head joint, (C) part affinity fields (output of second branch) for the association between head and neck joints, and (D) final human body pose estimation.

In Figure 6.18, the architecture used for the convolutional neural network is presented. As shown in the figure, the first part of the convolutional network consists in the first ten layers of the VGG-19 [117], which are used to perform a first analysis of the image, generating a set of feature maps, $\boldsymbol{F}$. Next, the network is split in three branches for the simultaneous learning of the heat maps, the part affinity fields, and the label detection.

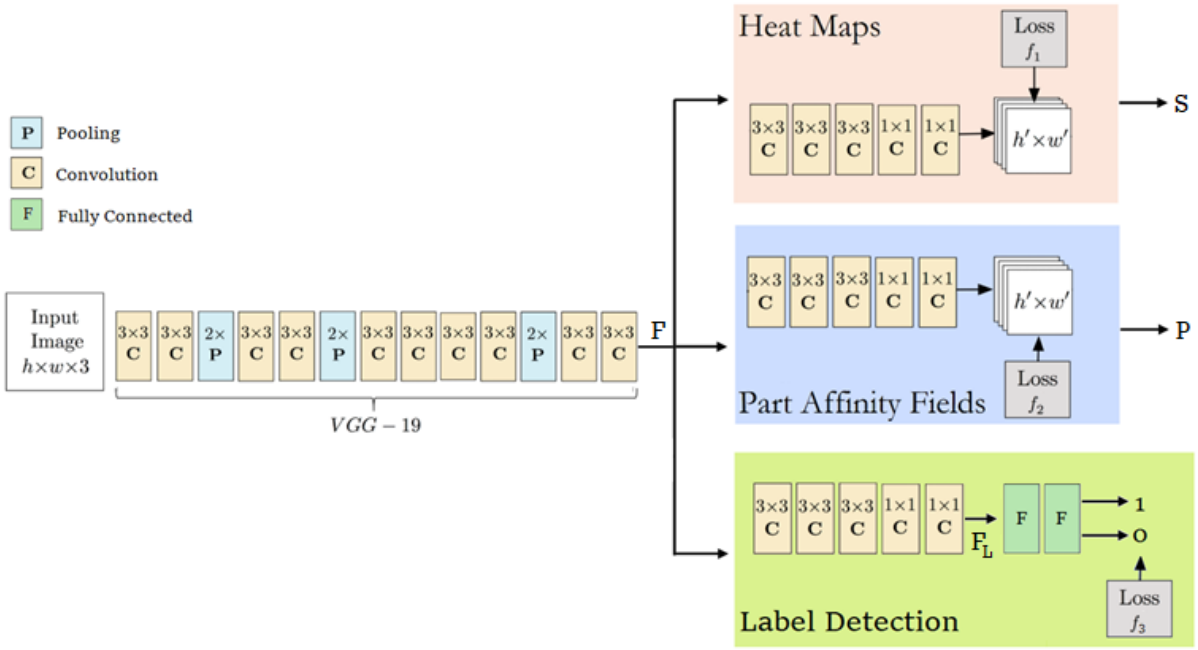


Figure 6.18: Architecture of the convolutional network used in the proposed method. The coral branch concerns the learning of the heat maps for body parts' detection and the blue branch concerns the part affinity fields for body part associations. Finally, the green branch is related to the label detection for joint categorization. Image adapted from [25].

Considering the use of the Caffe framework, the full network architecture was implemented in a prototxt file, $\mathbf{\Gamma^p}$ [115], where a lower level representation of the architecture can be seen in Figure 6.19, and its detailed layer dimensions in Table 6.2.

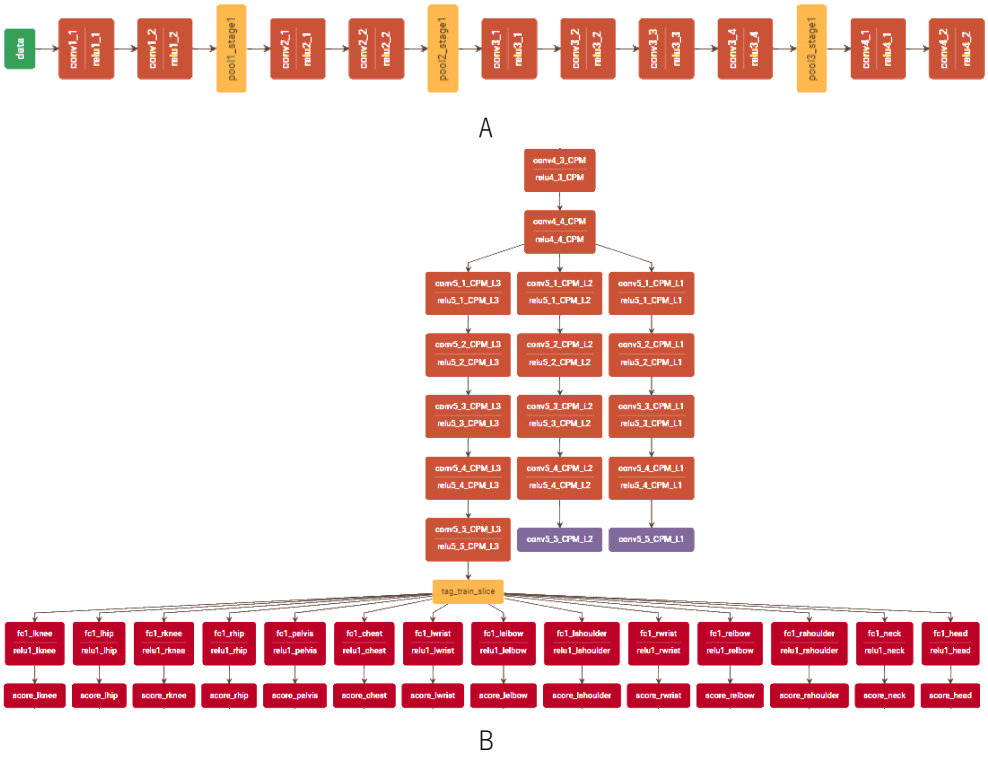


Figure 6.19: Architecture implementation in prototxt, $\mathbf{\Gamma^p}$, represented in lower level. (A) 10 layers of VGG19, and (B) stage one with three branches: heat maps ($L2$); part affinity fields ($L1$); and label detection ($L3$).

Table 6.2: Architecture implementation in prototxt, $\mathbf{\Gamma^p}$, where FC represents fully-connected, C represents convolution, and P represents pooling.

Name	Type	Size
VGG19		
conv1_#	C	3x3x64
pool1	P	2x2
conv2_#	C	3x3x128
pool2	P	2x2
conv3_#	C	3x3x256
pool3	P	2x2
conv4_1:2	C	3x3x512
Stage 1 Input		
conv4_3_CPM	C	3x3x256
conv4_4_CPM	C	3x3x128
Part Affinity Fields		
conv5_1:3_CPM_L1	C	3x3x128
conv5_4_CPM_L1	C	1x1x512
conv5_5_CPM_L1	C	1x1x26
Heat Maps		
conv5_1:3_CPM_L2	C	3x3x128
conv5_4_CPM_L2	C	1x1x512
conv5_5_CPM_L2	C	1x1x15
Label Detection		
conv5_1:3_CPM_L3	C	3x3x128
conv5_4_CPM_L3	C	1x1x256
conv5_5_CPM_L3	C	1x1x14
fc_#	FC	1x100
score_#	FC	1x2

6.2.1.1 Heat maps for body parts' detection

One of the branches of the convolutional network is used to predict confidence maps of each human body part (coral branch shown in Figure 6.18). As previously stated, a confidence map represents the belief that a body joint occurs in a given image pixel. In this sense, a confidence map can be seen as a 2D gaussian-like function, where the maximum of the gaussian map represents the ideal joint position. To train the method to predict heat maps, a loss function, $\boldsymbol{f_{heat}}$, was applied in the end of this branch to calculate the difference between predictions and ground-truth. In this case, the ground-truth for the confidence maps was generated by using manual labeling of the joint positions and constructing a gaussian map around the joint locations. The loss function for this branch is given by equation 6.7, where $\boldsymbol{nJ}$ represents the number of joints, $\boldsymbol{S_j}$ and $\boldsymbol{S_j^*}$ are the prediction and ground-truth maps for the part, $\boldsymbol{j}$, respectively ($\boldsymbol{S_j^* \equiv Pxy_j^C}$). In the test phase, the joint position for each body part is given by the maximum of the respective confidence map (i.e. its peak), after a non-maximum suppression.

$$f_1 = \sum_{j=1}^{nJ} \|S_j - S_j^*\| \tag{6.7}$$

6.2.1.2 Part affinity fields for body part association

To increase the accuracy of the body part detection, a second branch that measures the association between each pair of body parts is also included in the convolutional network (blue branch in Figure 6.18). This association is given by part affinity fields, which consists in a vector field between two body joints that encodes the direction between one body joint to another (Figure 6.17). The loss function associated to this branch is given by equation 6.8, where $\boldsymbol{pC}$ represents the number of connections betwen the different body parts, $\boldsymbol{P_c}$ is the prediction of the part affinity field and $\boldsymbol{P_c^*}$ is the ground-truth for the association.

$$f_2 = \sum_{c=1}^{pC} \|P_c - P_c^*\| \tag{6.8}$$

Besides refining the inference of the confidence maps during training, owing to the backpropagation scheme used during it, the part affinity fields are useful when there is more than one person in the image. Indeed, in this scenario, a confidence map by itself may not be enough for an accurate detection, because several peaks for the same joint may be detected (one for each person in the image). In this sense, the association between body parts are crucial to understand which joints belong to the same person. However, in this work, and given the FoV of the camera used, the focus was place in the detection of one person, and

therefore, part affinity fields was kept for its role in refining the heat map prediction.

6.2.1.3 Label detection for joint categorization

Owing to the limited size of an in-car environment and to the reduced FoV of the cameras used for monitoring in this environment, there is a higher probability of certain body joints being outside of the image, specially the extremity limbs (e.g. the driver can have its arm outside of the lateral window and the associated joint is therefore not present in the image). It is thus important to understand if the joint is present or not in the image to increase the accuracy of the human body pose estimation. To deal with this problem, a third branch was added to the network. This third branch allows to categorize the joint with a different label according to its existence in the image (i.e. the joint has the label 0 if it is outside of the image and label 1 if it is inside). This label detection was achieved by using a set of fully-connected layers to learn the non-linear combinations of the features, $\boldsymbol{F_L}$, extracted by the convolutional layers (green branch in Figure 6.18). Afterwards, a softmax layer was used to assign a probablity for the label detection, by taking the output of the fully-connected layers, logit, and transforming it into a vector with two prediction scores (one for each class $\boldsymbol{ci}$: present in the image or not). Each prediction score is given by $\boldsymbol{p_{ci,j}}$ in equation 6.9, where $\boldsymbol{y_{ci,j}}$ represents the output of the last fully-connected layer, logit, for class $\boldsymbol{ci}$, $\boldsymbol{K}$ corresponds to the number of classes (in this case $\boldsymbol{K = 2}$), and $\boldsymbol{nJ}$ represents the number of joints. Please note that the sum of the probabilities is equal to 1, and therefore, the joint's presence label is given by selecting the class with higher score (i.e. a one-hot encoding is used).

$$p_{ci,j} = \frac{e^{y_{ci,j}}}{\sum_{ci=1}^{K} e^{y_{ci,j}}}, ci = 1, ..., K \tag{6.9}$$

The cross entropy loss function is used for this branch, being given by equation 6.10, where $\boldsymbol{t_{ci,j}}$ is the ground-truth for the probability of each class (0 or 1).

$$f_3 = \sum_{j=1}^{J} -\frac{1}{K} \sum_{k=1}^{K} \log(p_{ci,j}, t_{ci,j}) \tag{6.10}$$

At test time, the prediciton of this third branch is used to verify which heat maps predicted in the first branch should be evaluated. If the label detection branch predict that the joint is not present in the image, it is considered not detected and the respective heat map is not evaluated. Otherwise, it is assumed that the joint is present in the image and its position is given by the maximum of the respective heat map, as stated in Section 6.2.1.1. The final human body pose estimation is obtained by combining the output of the three branches. Thus, the overall objective loss function is given by summing the loss of each branch (equation 6.11).

$$f = f_1 + f_2 + f_3 \tag{6.11}$$

6.2.2 Experiments

6.2.2.1 Dataset creation

Due to the inexistence of public datasets of depth images in an in-car scenario, it was needed to create a proprietary dataset, ($\boldsymbol{MD_{t'}}$ in equation 6.12). Owing to the deep learning nature of the method, a massive amount of data is needed as training data, and therefore, it was needed to create a large dataset. Moreover, besides the high number of training images, the training dataset must also be variable enough to include the large number of actions possible in this scenario. In fact, the accuracy of the method is very dependent of the quality of the dataset. In this sense, a dataset was constructed by acquiring depth images, $\boldsymbol{depth_{t',x,y}}$, using a ToF camera (Section 2.1.3.3) placed near the windshield in front of the driver. For the construction of the dataset, ten different cars, $\boldsymbol{Cr_{1:10}}$, were used to achieve the desired variability in terms of image background. For each one of the ten cars, five subjects acted as driver, $\boldsymbol{S_{1:5}}$, performing different actions, $\boldsymbol{A}$, inside a car (e.g. driving, putting the seat belt, picking up the phone, and others) to give the robustness needed for the dataset. The combination of the different cars and different subjects allowed to construct a dataset with 12200 depth images. The dataset was then divided in training, validation, and testing set with 8820, 1730, and 1650 images each, respectively. In this work, the training dataset was used to train the method, the validation set was used to test the progress of the performance of the method during training and to perform parameter tuning, and the testing set was used for the final validation of the proposed method. Note that the cars in each set of images differ from each other to achieve an unbiased evaluation. Concerning the ground-truth for the different body parts, it was constructed by manual labelling of the joint 2D positions, $\boldsymbol{(Pxyz_J^C)_{t'}}$, requiring a great amount of human resources and time (≈30 seconds per frame, totalising nearly 33 hours of labelling), Figure 6.20. Note that the performance of the method was only evaluated for these upper body joints once the lower body parts are naturally occluded when simulating a driving position. Moreover, the joint categorization (present or not) was also manually defined per image.

$$\therefore MD_{t'} = \{depth_{t',x,y}, (Pxyz_J^C)_{t'}\} \tag{6.12}$$

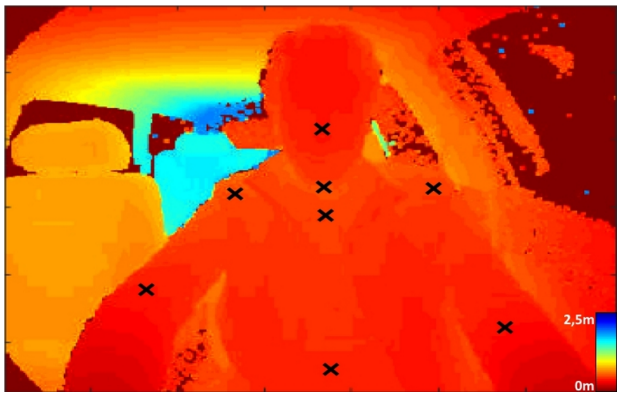


Figure 6.20: Real dataset with manual labelling (a black cross per each joint of interest).

6.2.2.2 Dataset augmentation

As above-mentioned, the ground-truth for the training dataset was obtained manually, a tedious and time-consuming task, which could represent a limitation in terms of the number of training images available. In this sense, besides the real dataset constructed, and as common in deep learning strategies, a data augmentation layer was implemented, allowing to generate more training images than the ones initally labelled. Such data augmentation strategy allows to increase the variability of the training dataset. Traditionally, data augmentation strategies rely on image flip, rotation, and scaling. Although it is an effective way of increasing image variability during training, such strategies do not modify the depth information of the image. Although such feature is not so problematic for RGB images, it can be for depth ones, as changing the depth of these images can be a useful way to simulate different camera positions in the real world (i.e. the distance between the camera and objects). In this work, the implemented data augmentation layer (CPM_DA) can simulate these changes in terms of camera's position, allowing to create images where the objects (i.e. the driver) are closer or farther from the camera than they were in fact. In Algorithm 6.2.1 the first step for the data augmentation approach is to read the input frame and associated ground-truth that is being feed to the training pipeline, as well as the 3D data augmentation specific hiper-parameters that were inserted in the prototxt file, $\mathbf{\Gamma^p}$, allowing for easy customizations for each training. Afterwards, the 2.5 depth image (and associated ground-truth) is converted in a 3D point cloud, using the intrinsic parameters of the camera (equation 6.13). Upon obtaining the 3D camera coordinates, all points can be transformed by applying a given translation, moving the 3D point cloud in any direction and in any axis.

$$Pxyz = (Pxy)^{\intercal} \cdot IM_C \tag{6.13}$$

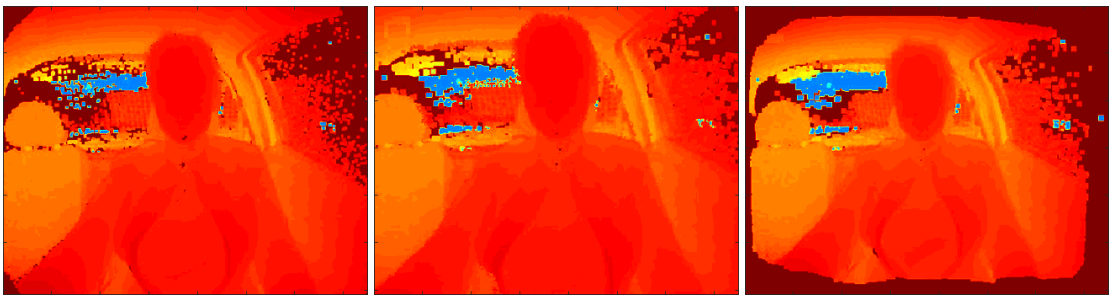
The final step consists in using the intrinsic camera parameters (equation 4.4) to transform the translated

point cloud into a new 2.5D depth image (equation 6.14). In Figure 6.21, it is possible to visualize two examples of the result of our data augmentation strategy: one simulates the camera closer to the driver (Figure 6.21B) and the other simulates the camera located farther from the driver (Figure 6.21C).

$$Pxy = Pxyz \cdot IM_C \tag{6.14}$$

Algorithm 6.2.1 PAF CPM_DA

initialize:
Get the depth and body pose sample from cpm_data_transform (CPM_DA)
$depth_{x,y} \leftarrow CPM\_DA$ (**Figure 6.21A**)
$Pxy_J^C \leftarrow CPM\_DA$
Get the scalling probabilities
$\sigma_x \leftarrow \Gamma^p$
$\sigma_y \leftarrow \Gamma^p$
$\sigma_z \leftarrow \Gamma^p$
$P_{augment} \leftarrow \Gamma^p$
$Px_{augment} \leftarrow \Gamma^p$
$Py_{augment} \leftarrow \Gamma^p$
$Pz_{augment} \leftarrow \Gamma^p$
# Check if there is need to augment
if $P_{augment} \geq (X \sim \mathcal{U}(0,1))$ **then**
# Get new transformation
  if $Px_{augment} \geq (X \sim \mathcal{U}(0,1))$ **then**
    $tr_x \leftarrow \mathcal{U}(0, \sigma_x)$
  end if
  if $Py_{augment} \geq (X \sim \mathcal{U}(0,1))$ **then**
    $tr_y \leftarrow \mathcal{U}(0, \sigma_y)$
  end if
  if $Pz_{augment} \geq (X \sim \mathcal{U}(0,1))$ **then**
    $tr_z \leftarrow \mathcal{U}(0, \sigma_z)$
  end if
# Convert 2D coordinates to 3D
  $depth_{x,y,z} \leftarrow Eq.6.13(depth_{x,y})$
  $Pxyz_J^C \leftarrow Eq.6.13(Pxy_J^C)$
# Apply 3D transformation to the sample
  for each u, v, w in x, y, z **do**
    $\{x, y, z\} \leftarrow depth_{u,v,w}$
    $depth_{u,v,w} \leftarrow \{x + tr_x, y + tr_y, z + tr_z\}$
  end for
  for each j in J **do**
    $\{x, y, z\} \leftarrow Pxyz_J^C$
    $Pxyz_J^C \leftarrow \{x + tr_x, y + tr_y, z + tr_z\}$
  end for
# Convert 3D coordinates to 2D
  $depth_{x,y} \leftarrow$ **Eq.6.14(**$\boldsymbol{depth_{x,y,z}}$**)** (**Figures 6.21B and 6.21C**)
  $Pxy_J^C \leftarrow$ **Eq.6.14(**$Pxyz_J^C$**)**
end if



A B C

Figure 6.21: Data augmentation strategy. (A) original depth image, (B) augmented image that simulates the positioning of the camera closer to the driver, and (C) augmented image that simulates the camera being located farther from the driver.

6.2.2.3 Implementation details

Regarding all customizations that were mentioned, the definition and optimization of 3D augmentation parameters, and label detection branch layers, required several evaluations. Other deep learning training hiper-parameters were also optimized, these are fundamental and can have a significant importance in the final accuracy of the method. Concerning the learning rate, this parameter was experimentally set to 0.0004. For the model optimization, the Adam solver [118] was used with a regularization term of 0.01. Note that these parameters were chosen by evaluating the method in the validation dataset.

3D data augmentation optimization (AE1):

To experimentally define the hiper-parameters inserted onto the prototxt, $\boldsymbol{\Gamma^p}$, four sub-evaluations were defined, for each the hiper-parameters were changed (Table 6.3) and the joint distance error, $\boldsymbol{mAD}$, was evaluated (Table 6.4). Results from $\boldsymbol{AE1:4}$ defined the hiper-parameters used for the CPM_DA algorithm (Algorithm 6.2.1).

Table 6.3: AE1 sub-evaluation hiper-parameter configurations: $\boldsymbol{P_{augment}}$ probability of 3D augmenting (i.e. changing the input image); $\boldsymbol{Px_{augment}}$ probability of translating the camera in the X axis; $\boldsymbol{Py_{augment}}$ probability of translating the camera in the Y axis; $\boldsymbol{Pz_{augment}}$ probability of translating the camera in the Z axis; $\boldsymbol{\sigma_x}$ standard deviation for the translation in the X axis; $\boldsymbol{\sigma_y}$ standard deviation for the translation in the Y axis; and $\boldsymbol{\sigma_z}$ standard deviation for the translation in the Z axis.

Evaluation	$\boldsymbol{P_{augment}}$	$\boldsymbol{Px_{augment}}$	$\boldsymbol{Py_{augment}}$	$\boldsymbol{Pz_{augment}}$	$\boldsymbol{\sigma_x}$	$\boldsymbol{\sigma_y}$	$\boldsymbol{\sigma_z}$
$AE1:1$	0						
$AE1:2$	1	0.1	0.1	0.8	0.05	0.05	0.05
$AE1:3$	0.8	0.1	0.1	0.8	0.05	0.05	0.05
$AE1:4$	0.8	0.1	0.1	0.8	0.15	0.15	0.15

Table 6.4: AE1 sub-evaluation performance in the evaluation dataset, assessed in terms of distance error ($\boldsymbol{mAD}$, pixels).

Evaluation	H	N	RS	RE	RW	LS	LE	LW	C	P	RH	LH	Mean
$AE1:1$	8.00	6.08	9.06	8.06	9.85	8.00	8.06	15.26	11.05	13.04	13.89	20.01	10.44
$AE1:2$	7.28	5.83	7.21	7.81	11.70	6.32	7.81	16.12	10.44	10.44	12.77	21.38	9.91
$AE1:3$	7.81	6.00	7.62	7.62	10.44	7.00	8.06	15.81	10.05	10.30	12.17	17.69	9.55
$AE1:4$	7.07	6.00	7.21	7.21	10.44	6.40	7.81	15.52	10.00	11.18	12.37	16.40	9.34

Third branch layer optimization (AE2):

To optimize the third branch defined onto the prototxt, $\boldsymbol{\Gamma^p}$, three sub-evaluations were defined and the joint distance error $\boldsymbol{mAD}$ was evaluated. (AE2:1) uses the algorithm without third branch, (AE2:2) uses the algorithm with third branch (Table 6.5) with three labels (visible, occluded, and outside image), (AE2:3) uses the final algorithm arquitecture shown in Table 6.2. Results from $\boldsymbol{AE2:3}$ defined the final algorithmic architecture (Table 6.6).

Table 6.5: AE2:2 third branch architecture implementation in prototxt, $\boldsymbol{\Gamma^p}$, where FC represents fully-connected, C represents convolution, and P represents pooling.

Label Detection		
conv5_1:3_CPM_L3	C	3x3x128
conv5_4_CPM_L3	C	1x1x256
conv5_5_CPM_L3	C	1x1x14
fc_#	FC	1x100
score_#	FC	1x3

Table 6.6: AE2 sub-evaluation performance in the evaluation dataset, assessed in terms of distance error ($\boldsymbol{mAD}$, pixels).

Evaluation	H	N	RS	RE	RW	LS	LE	LW	C	P	RH	LH	Mean
$AE2:1$	8.00	6.08	9.06	8.06	9.85	8.00	8.06	15.26	11.05	13.04	13.89	20.01	10.44
$AE2:2$	7.28	5.83	7.21	7.81	11.70	6.32	7.81	16.12	10.44	10.44	12.77	21.38	9.91
$AE2:3$	7.81	6.00	7.62	7.62	10.44	7.00	8.06	15.81	10.05	10.30	12.17	17.69	9.55

6.2.3 Results

One important task to correctly implement a deep learning strategy is to evaluate the progress of the training in the validation dataset. Besides being useful to conduct experiments related with the best parameters to be used in the deep learning strategy, the validation dataset is also needed to detect problems like overfitting to the training data, which may cause failure of the method when applied in a different set of images. In this sense, evaluating the method's performance during training in the validation dataset allows to detect when the training converges, avoiding overfitting problems. Figure 6.22 presents an example graph showing the progression of the loss during training in both training and validation datasets.

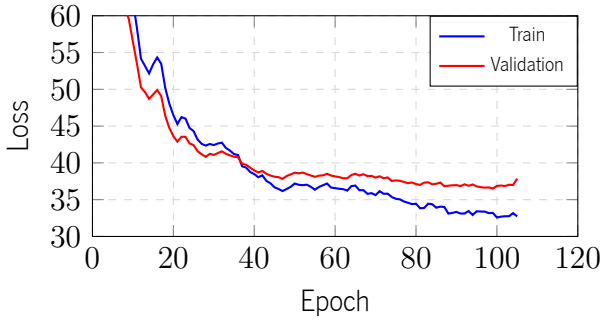


Figure 6.22: Method's performance (loss in function of epochs) during training in the training (blue line) and validation (red line) datasets.

To evaluate the performance of the proposed method in the testing set, PCKh measure (in pixels, using a matching threshold given by 50% of the head segment length) and the Area Under Curve (AUC) were used as metrics [103]. Table 6.7 summarizes the average results for the full body, and individual joints.

Table 6.7: Evaluation results (PCKh@0.5) per joint group.

Head[1]	**Shoulder**[2]	**Elbow**[3]	**Wrist**[4]	**Hip**[5]	**Total**[6]	**AUC**[6]
99.35	99.32	97.91	87.57	98.17	97.58	76.75

[1] Head uses head and neck joints; [2] Shoulder uses rshoulder, lshoulder and chest joints.

[3] Elbow uses relbow and lelbow joints; [4] Wrist uses rwrist and lwrist joints.

[5] Hip uses rhip, lhip and pelvis joints.

[6] does matching threshold to $26\ pixels$ (empirical extraction of head segment size).

Figure 6.23 presents the PCKh@0.5 values for the full and individual body parts. Moreover, the results shown in the table were achieved by the model obtained in the ideal epoch for early training stopping, which were estimated using Figure 6.22.

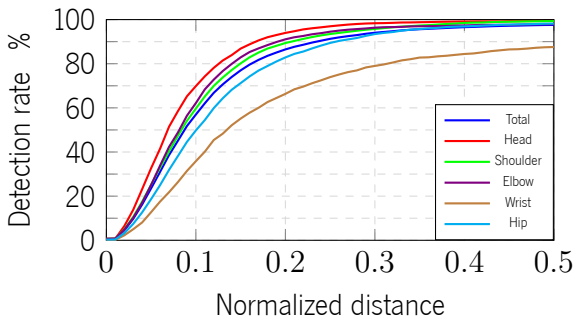


Figure 6.23: PCKh total and per joint groups.

Finally, in Figure 6.24, some example results of the proposed strategy for human body pose estimation are presented.

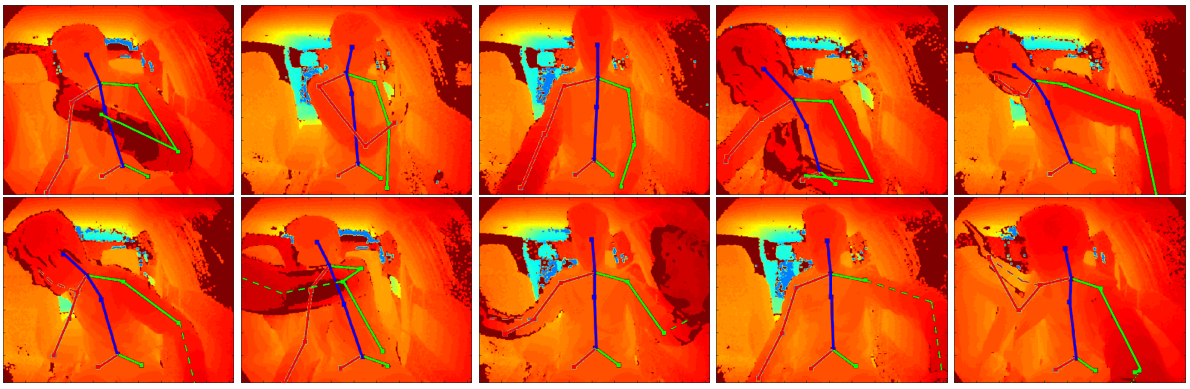


Figure 6.24: Qualitative results of the proposed human body pose detection method. The first row presents examples of good results. In the second row, some examples where the pose estimation failed for a few joints are illustrated, with the ground-truth pose in dashed lines.

6.2.4 Discussion

This chapter proposed a method for human body pose estimation in an in-car scenario. As stated, an important study to be performed during the implementation of a deep learning strategy concerns the evolution of the training. For that, its performance during training must be evaluated in the training and validation datasets. Analyzing Figure 6.22, it is possible to visualize that the ideal timing for stopping the training would be approximately around the 60^{th} epoch. After this point of the training, the graph suggests that the model may start suffering from overfit to the training dataset, resulting in a very good performance for the images presented in this dataset but a lower or equal performance in the validation dataset. After analyzing the ideal epoch for early stopping, the model obtained in this epoch was used for the final validation of the method in the testing set. Analyzing Table 6.7, it is possible to verify the good performance of the human body pose detection method, with a PCKh@0.5 total and AUC of 97.58% and 76.75% respectively. The worst results

were obtained for the wrist joints. This can be explained by the fact that these joints are frequenty near the image's limits, which can lead to a lower accuracy of the deep learning strategy. Moreover, owing to the proximity of the camera to the driver, these joints are not always present in the camera's FoV, which hampers the training process for these joints. This less accurate detection for these extremity joints can also be visualized in Figure 6.23. Concerning the computational time required by the method, the proposed human body pose detection methods takes approximately 60 milliseconds per image, which gives a performance of 16.7 fps, which proves the nearly real time capability of the method, as well as, the compatibility with the ToF sensor use-case 3 (Table 2.3). Note that this runtime analysis was performed using a NVIDIA GeForce GTX-1070 Graphics Processing Unit (GPU). Overall, the obtained results suggest the feasibility of using the proposed method for monitoring passengers inside a car. The output of the proposed method, which is the joint positions, allows to generate the human body pose for the driver or other passengers, which can give important information such as who is inside the vehicle or its activity (i.e. what the passenger is doing). The proposed method, when combined with other post-processing techniques, may thus be used for monitoring applications, which can be useful for safety issues. Some possible use cases follow. For example, it is possible to use the information of the human body pose to automatically suppress the airbags of the car in the presence of a child. Moreover, it is also possible to use the human body pose to estimate the time that the person will take to pass from an automatic driving to a manual driving, allowing to generate alerts related to this issue. Concerning the driver's activity, this information is useful for detecting, for example, violent actions, which is very important in a Shared Autonomous Vehicles (SAV) context. One important aspect to take into account in the proposed method is its application in a real setup. In fact, to achieve a real time estimation of the human body pose inside a car, a high computational power is needed, which can be a limitation for an in-car scenario. However, the constant growth in computation capability of the technology already allows the application of the proposed method in a real scenario. Another aspect to take into account is that the application of software in autonomous cars should follow some existing standards, in order to ensure its applicability in a real scenario. Nevertheless, all these issues will be addressed in the future to achieve a technical solution of application of the proposed method in a real scenario.

6.3 Conclusions and Future Work

This chapter presented a framework to detect the human body pose in depth images acquired inside a car. This method consists in a evaluated/selected deep learning strategy where a new convolutional network configuration with three branches was used for simultaneous learning of confidence maps for each joint po-

sition, body parts' associations, and joint categorization (regarding its existence in the image). The proposed framework was validated in 1650 depth images, achieving a PCKh@0.5 total and AUC of 97.58% and 76.75% respectively. Overall, the proposed human body pose estimation method proved to be successful and accurate to detect the driver's pose, while showing a nearly real time performance. In future work, there will be an extension of the human body pose estimation to more than one person, allowing to monitor not only the driver but all the car occupants. In addition, one intends to modify it into inferring the 3D joint position, therefore better exploiting the 3D information provided by the depth image.

Chapter 7

Algorithm Evaluation

This chapter is sequential to Chapter 6 in terms of development, and is the culmination of the entire development pipeline shown in Chapter 1. This chapter shows the final evaluation of the selected/customized algorithm from Chapter 6 with datasets from chapters 4 and 5 to ultimately improve its accuracy and generalization ability in regard to Time-of-Flight (ToF) placement inside the vehicle, subjects and actions.

In the first section of this chapter, a toolchain for algorithmic evaluation is shown. This toolchain allows for two evaluation modes: (1) qualitative evaluation through online ToF interface; and (2) quantitative evaluation through offline dataset (i.e. $\boldsymbol{RD_{t'}}$,$\boldsymbol{SD_{t'}}$) interface.

In the second section of this chapter, the customized Part Affinity Fields (PAF) algorithm is evaluated in three ToF Point-of-View (PoV) scenarios, through a combination of datasets.

Contents

7.1 Evaluation Toolchain

7.1.1 Toolchain Overview

A toolchain was created to evaluate the customized algorithm proposed in Chapter 6, with two modes: (1) qualitative evaluation through online mode, where it is possible to interface with the ToF sensor and the algorithm, showing its inference ability with respect to the ToF PoV; and (2) quantitative evaluation through offline mode, where it is possible to interface with the algorithm and any type of datasets from chapters 4 and 5 to ultimately compute evaluation metrics (as defined in Section 6.1.2.2). Considering the fact that the toolchain does not require real time access of multiple systems, the entire implementation was done in MATLAB. Figures 7.1 and 7.2 present a pipeline overview of the two evaluation modes implemented onto the toolchain.

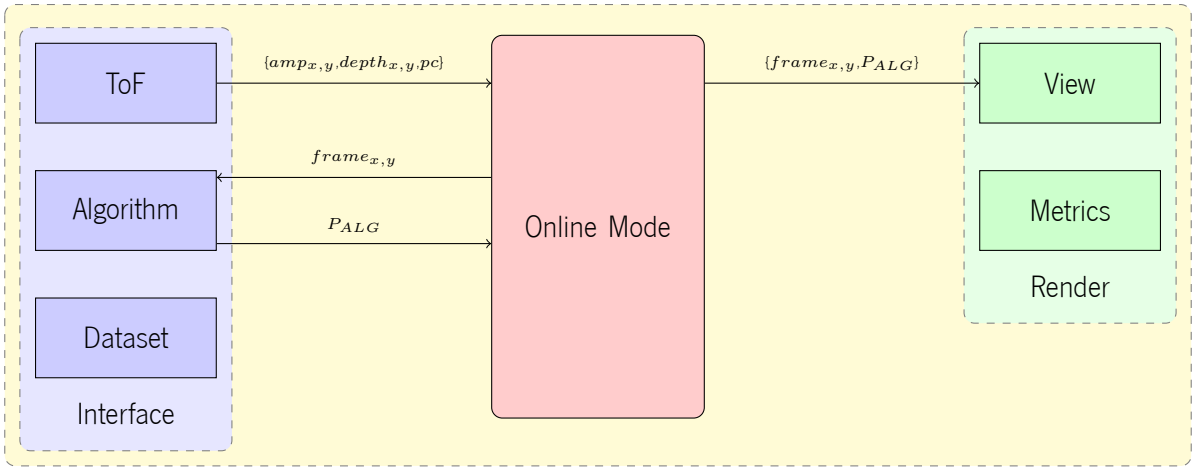


Figure 7.1: Algorithmic evaluation toolchain pipeline for online mode.

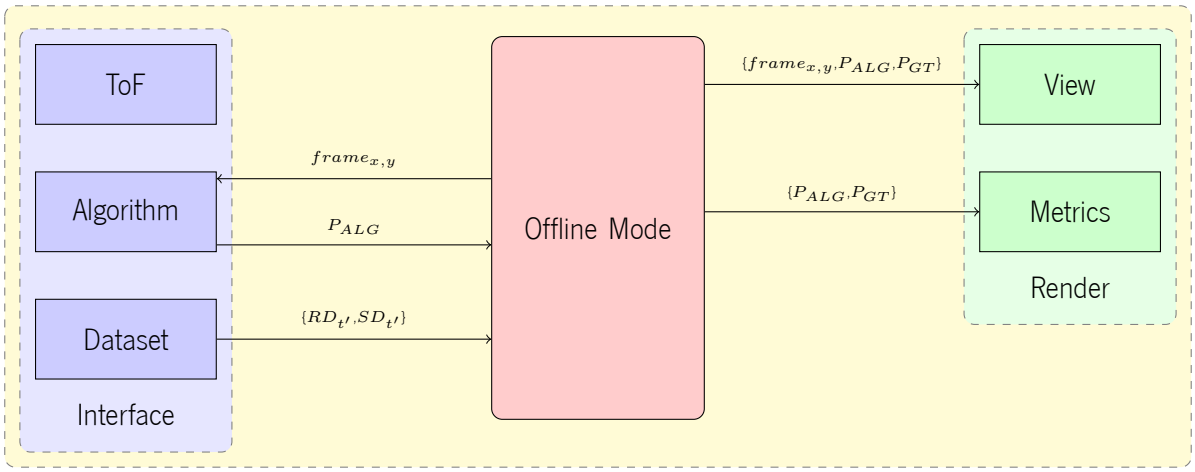


Figure 7.2: Algorithmic evaluation toolchain pipeline for offline mode.

7.1.1.1 Interface

The first conceptual module from the toolchain is the interface, through it the toolchain as access to the ToF, algorithm, and datasets.

ToF (Section 2.1.3.3) interface is done with the provided SDK (Algorithm 2.1.5), allowing for user-defined, $\mathbf{\Gamma^u}$ use case (Table 2.3) and frame selection, $\{\boldsymbol{amp_{t',x,y}}, \boldsymbol{depth_{t',x,y}}, \boldsymbol{pc_{t',f}}\}$. Algorithm interface is done through Interprocess Communications (IC) [119], to achieve algorithmic language implementation abstraction. A normalized frame is encoded (i.e. $\boldsymbol{frame_{x,y}} \equiv \{\boldsymbol{amp_{x,y}}, \boldsymbol{depth_{x,y}}, \boldsymbol{pc_f}\}$) and sent from the toolchain into the algorithm, which then infers the human body pose (i.e. 2D $\boldsymbol{Pxy_J^C}$, or 3D $\boldsymbol{Pxyz_J^C}$, represented by $\boldsymbol{P_{ALG}}$) and sends it back to the toolchain. Dataset interface is done through file IO, where any type of dataset is able to be imported (i.e. $\boldsymbol{RD_{t'}}, \boldsymbol{SD_{t'}}$).

7.1.1.2 Online Mode (Qualitative)

In online mode, the toolchain is cyclically sending a user selected frame, $\boldsymbol{frame_{x,y}}$, which is normalized ($\boldsymbol{amp_{x,y}}$ $[0; 500] \equiv [0; 255]$, $\boldsymbol{depth_{x,y}}$ $[0; 1.8]$ $m \equiv [0; 255]$) before being sent to the algorithm through IC. After this interaction, the toolchain receives an inferred 2D body pose (i.e. $\boldsymbol{P_{ALG}} \equiv \boldsymbol{Pxy_J^C}$) through IC.

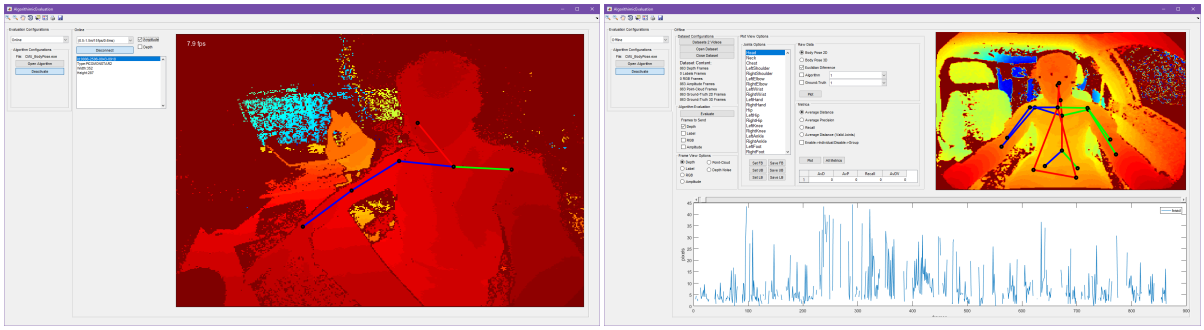
7.1.1.3 Offline Mode (Quantitative)

In offline mode, the toolchain imports a full dataset, $\{\boldsymbol{RD_{t'}}, \boldsymbol{SD_{t'}}\}$, normalizes a set of user selected frames from the available dataset, $\boldsymbol{frame_{x,y}}$, (i.e. $\boldsymbol{amp_{x,y}}$, $[0; 500] \equiv [0; 255]$; $\boldsymbol{depth_{x,y}}, \boldsymbol{nstdepth_{x,y}}$, $[0; 1.8]$ $m \equiv [0; 255]$; $\boldsymbol{pc_f}$, pcx and pcy with $[-1.5; 1.5]$ $m \equiv [0; 255]$, and pcz with $[0; 1.8]$ $m \equiv [0; 255]$) and sends them to the algorithm through IC. After each cycle, the toolchain receives an inferred 2D body pose (i.e. $\boldsymbol{P_{ALG}} \equiv \{\boldsymbol{Pxy_J^C}, \boldsymbol{Pxyz_J^C}\}$) through IC that is stacked onto a sequence of body poses, $\{(\boldsymbol{Pxyz_J^C})_{t'}, (\boldsymbol{Pxyz_J^C})_{t'}\}$. After the entire cycle of interaction between toolchain and algorithm, the toolchain is able to calculate the metrics specified in Section 6.1.2.2 by comparing the inferred pose with the ground-truth (i.e. for 2D, $\boldsymbol{P_{ALG}} \equiv \boldsymbol{Pxy_J^C}$, $\boldsymbol{P_{GT}} \equiv \boldsymbol{Pxy_J^C}$, $\boldsymbol{t' = 0, ..., T}$).

7.1.1.4 Render

A Graphical User Interface (GUI) is used to render the user selected frames in online (i.e. $\boldsymbol{amp_{x,y}}, \boldsymbol{depth_{x,y}}$) and offline (i.e. $\boldsymbol{amp_{x,y}}, \boldsymbol{depth_{x,y}}, \boldsymbol{labels_{x,y}}, \boldsymbol{noisedepth_{x,y}}, \boldsymbol{nstdepth_{x,y}}, \boldsymbol{RGB_{x,y}}, \boldsymbol{pc_f}$) modes with overlaid dataset and algorithmic inferred human body pose, $\boldsymbol{Pxy_J^C}$, $\boldsymbol{Pxyz_J^C}$. An extra rendering function is available for the offline mode, being possible to calculate (i.e. $\boldsymbol{mAD}, \boldsymbol{mAD^{\Gamma^t}}, \boldsymbol{mAP^{\Gamma^t}}, \boldsymbol{mAR}$, where

$\boldsymbol{\Gamma^t}$ is a user-defined threshold) and plot, $\boldsymbol{d_{t',j}}$, specific metrics. Figures 7.3A and 7.3B show the rendering capabilities mentioned.



A B

Figure 7.3: Algorithmic evaluation toolchain: (A) online evaluation, with ToF and algorithmic interface allowing for qualitative evaluation; and (B) offline evaluation, with dataset and algorithmic interface allowing for quantitative evaluation.

7.2 Experiments

To consolidate the entire development pipeline presented in Chapter 1, the algorithm from Chapter 6 needs to be trained with a large dataset comprised by samples generated by the toolchains in chapter 4 and 5. In this case, the single experimental scenario considered was the 2D pose estimation from point cloud, which had presented better results in previous experiments (chapters 4 and 5).

7.2.1 Evaluation Data

In this sense, four previously generated datasets were used, namely two real datasets (generated with the toolchain presented in Chapter 4): the MoLa R89k InCar Dataset (i.e. an extension to the MoLa R8.7k InCar dataset [102]); and the MoLa R10k InCar Dataset (which we made public) [108]; plus two synthetic datasets (generated with the toolchain presented in Chapter 5): the MoLa S25k InCar Dataset (which we made public) [109]; and the MoLa S63k InCar Dataset.

($\boldsymbol{RD^1_{t'}}$) MoLa R89k InCar Dataset consists in five recorded subjects, $\boldsymbol{S_{1:5}}$, nine redundant actions, $\boldsymbol{A_{1:9}}$, for subjects $\boldsymbol{S_{1:4}}$ and eleven, $\boldsymbol{A_{1:11}}$, for subject $\boldsymbol{S_5}$, two car seat positions, $\boldsymbol{P_{1:2}}$, (i.e. front passenger and driver), with a front row PoV, totaling 89079 samples. ($\boldsymbol{RD^2_{t'}}$) MoLa R10k InCar Dataset consists in three recorded subjects, $\boldsymbol{S_{1:3}}$, two redundant actions, $\boldsymbol{A_{1:2}}$, each, with a driver PoV, totaling 10482 samples. ($\boldsymbol{SD^1_{t'}}$) MoLa S25k InCar Dataset consists in seven car models, $\boldsymbol{CM_{1:7}}$, and eighteen subjects, $\boldsymbol{Z_{1:18}}$, with associated Gaussian poses, $\boldsymbol{N_{hgp}}$, with a driver PoV, totalling 25000 samples. ($\boldsymbol{SD^2_{t'}}$) MoLa S63k InCar

Dataset consists in seven car models, $\boldsymbol{CM_{1:7}}$, and eighteen subjects, $\boldsymbol{Z_{1:18}}$, with associated Gaussian poses, $\boldsymbol{N_{hgp}}$, two car seat positions, $\boldsymbol{P_{1:2}}$, (i.e. front passenger and driver), with a front row PoV, totalling 63000 samples.

Both datasets are identical (i.e. $\boldsymbol{RD_{t'} \equiv SD_{t'}}$) in terms of sample data types for depth frame ($\boldsymbol{depth_{t',x,y} \equiv nstdepth_{t',x,y}}$), point cloud ($\boldsymbol{pc_{t',f}}$), 2D and 3D body pose ($\boldsymbol{(Pxy_J^C)_{t'}}$,$\boldsymbol{(Pxyz_J^C)_{t'}}$), giving us the opportunity to evaluate the experimental scenario that presented best results in chapters 4 and 5, (i.e. the 2D pose estimation from point cloud [section 4.2.2.1 and 5.3.2.3]). To assess the algorithm's inference generalization capacities, the training $\boldsymbol{FT}$, validation $\boldsymbol{FE}$ (Table 7.1) and evaluation $\boldsymbol{FM}$ (Table 7.2) sets were divided in three distinct groups: one with ToF front row PoV, one with ToF driver PoV, and one with both.

Table 7.1: Validation sets. ($\boldsymbol{FE1}$) samples with ToF front row PoV; ($\boldsymbol{FE2}$) samples with ToF driver PoV; and ($\boldsymbol{FE3}$) samples with ToF front row and driver PoV.

	$\boldsymbol{RD_{t'}^1}$		$\boldsymbol{RD_{t'}^2}$		
Validation	**Setup**	**Samples**	**Setup**	**Samples**	**Total**
$FE1$	$S_5, A_{10:11}, P_{1:2}$	3430			3430
$FE2$			S_3, A_2	892	892
$FE3$	$S_5, A_{10:11}, P_{1:2}$	3430	S_3, A_2	892	4322

Table 7.2: Evaluation sets. ($\boldsymbol{FM1}$) estimation with ToF front row PoV; ($\boldsymbol{FM2}$) estimation with ToF driver PoV; and ($\boldsymbol{FM3}$) estimation with ToF front row and driver PoV.

	$\boldsymbol{RD_{t'}^1}$		$\boldsymbol{RD_{t'}^2}$		
Evaluation	**Setup**	**Samples**	**Setup**	**Samples**	**Total**
$FM1$	$S_2, A_{1:9}, P_{1:2}$	8763			8763
$FM2$			S_3, A_1	2644	2644
$FM3$	$S_2, A_{1:9}, P_1$	8763	S_3, A_1	2644	11407

Considering the three groups, the training sets were divided in six sets (Table 7.3), $\boldsymbol{FT}$, to find the best real/synthetic dataset combination that achieves highest accuracy and generalization. The first three training combination, $\boldsymbol{FT_{1:3}}$, focus on the usage of samples from real datasets $\boldsymbol{RD_{t'}}$: ($\boldsymbol{FT1}$) training with $\boldsymbol{RD_{t'}^1}$; ($\boldsymbol{FT2}$) training with $\boldsymbol{RD_{t'}^2}$; and ($\boldsymbol{FT3}$) training with $\boldsymbol{RD_{t'}^1}$ and $\boldsymbol{RD_{t'}^2}$. The last three training configurations, $\boldsymbol{FT3_{1:3}}$, focus on the usage of samples from synthetic datasets $\boldsymbol{SD_{t'}}$ on top of the real ones: ($\boldsymbol{FT3_1}$)

training with $\boldsymbol{RD^1_{t'}}$, $\boldsymbol{RD^2_{t'}}$ and $\boldsymbol{SD^1_{t'}}$; ($\boldsymbol{FT3_2}$) training with $\boldsymbol{RD^1_{t'}}$, $\boldsymbol{RD^2_{t'}}$ and $\boldsymbol{SD^2_{t'}}$; and ($\boldsymbol{FT3_3}$) training with $\boldsymbol{RD^1_{t'}}$, $\boldsymbol{RD^2_{t'}}$, $\boldsymbol{SD^1_{t'}}$ and $\boldsymbol{SD^2_{t'}}$.

Table 7.3: Training sets. ($\boldsymbol{FT1}$) training with $\boldsymbol{RD^1_{t'}}$; ($\boldsymbol{FT2}$) training with $\boldsymbol{RD^2_{t'}}$; ($\boldsymbol{FT3}$) training with $\boldsymbol{RD^1_{t'}}$ and $\boldsymbol{RD^2_{t'}}$; ($\boldsymbol{FT3_1}$) training with $\boldsymbol{RD^1_{t'}}$, $\boldsymbol{RD^2_{t'}}$ and $\boldsymbol{SD^1_{t'}}$; ($\boldsymbol{FT3_2}$) training with $\boldsymbol{RD^1_{t'}}$, $\boldsymbol{RD^2_{t'}}$ and $\boldsymbol{SD^2_{t'}}$; and ($\boldsymbol{FT3_3}$) training with $\boldsymbol{RD^1_{t'}}$, $\boldsymbol{RD^2_{t'}}$, $\boldsymbol{SD^1_{t'}}$ and $\boldsymbol{SD^2_{t'}}$.

	$RD^1_{t'}$		$RD^2_{t'}$		$SD^1_{t'}$		$SD^2_{t'}$		
Training	**Setup**	**Samples**	**Setup**	**Samples**	**Setup**	**Samples**	**Setup**	**Samples**	**Total**
$FT1^1$	$S_{1,3:5}, A_{1:9}, P_{1:2}$	76886							76886
$FT2^2$			$S_{1:2}, A_{1:2}$	6946					6946
$FT3^3$	$S_{1:5}, A_{1:9}, P_{1:2}$	76886	$S_{1:2}, A_{1:2}$	6946					83832
$FT3^3_1$	$S_{1:5}, A_{1:9}, P_{1:2}$	76886	$S_{1:2}, A_{1:2}$	6946	$CM_{1:7}, Z_{1:18}, N_{hgp}$	25200			109032
$FT3^3_2$	$S_{1:5}, A_{1:9}, P_{1:2}$	76886	$S_{1:2}, A_{1:2}$	6946			$CM_{1:7}, Z_{1:18}, N_{hgp}, P_{1:2}$	63000	146832
$FT3^1_3$	$S_{1:5}, A_{1:9}, P_{1:2}$	76886	$S_{1:2}, A_{1:2}$	6946	$CM_{1:7}, Z_{1:18}, N_{hgp}$	25200	$CM_{1:7}, Z_{1:18}, N_{hgp}, P_{1:2}$	63000	172032

Best epoch is chosen with different validation sets (Table 7.1): [1] $FE1$; [2] $FE2$; and [3] $FE3$

Finally, the algorithm was trained in each training set and evaluated in all evaluation sets, therefore being able to check the algorithm's generalization ability. To avoid misleading evaluation results, each configuration of data (i.e. subjects, actions, positions and cars) was only considered in one of the sets, i.e. either training, validation or evaluation sets.

7.2.2 2D Pose Estimation from point cloud

Similar to section 4.2.2.1 and 5.3.2.3, we evaluate the 2D pose estimation accuracy from 3D point cloud. Table 7.4 summarizes the average results for the full body with three ToF sensor placement scenarios, $\boldsymbol{FM\#}$, with the results for individual joints being presented in Table 7.5. Figures 7.4 and 7.5 present the PCKh@0.5 values for the full body and body parts respectively, for each ToF PoV scenario, $\boldsymbol{FM\#}$.

Table 7.4: PCKh measure and AUC values averaged over all 14 joints, for all training sets, ***FT#***, in each ToF PoV experimental scenarios, ***FM#***: ToF front row PoV (FM1); ToF driver PoV (FM2); and ToF front row and driver PoV (FM3).

		$FT1$	$FT2$	$FT3$	$FT3_1$	$FT3_2$	$FT3_3$
FM1	PCKh[1]	93.14	0.48	93.15	92.61	94.66	94.59
	AUC	59.87	0.13	58.57	61.00	61.58	61.78
FM2	PCKh[1]	73.68	91.80	95.45	94.22	94.87	96.29
	AUC	43.46	58.58	80.21	77.99	77.59	80.45
FM3	PCKh[1]	88.74	21.11	93.67	92.97	94.71	94.97
	AUC	59.16	11.33	63.46	64.84	65.20	65.99

[1] $FM\#$ does matching threshold to $26\ pixels$

Table 7.5: Evaluation results (PCKh@0.5) per joint group for all training sets, ***FT#***, in each ToF PoV experimental scenarios, ***FM#***: ToF front row PoV (FM1); ToF driver PoV (FM2); and ToF front row and driver PoV (FM3).

		$FT1$	$FT2$	$FT3$	$FT3_1$	$FT3_2$	$FT3_3$
FM1[6]	Head[1]	99.16	0.04	99.04	99.21	99.41	99.94
	Shoulder[2]	99.57	1.77	99.49	99.59	99.59	99.51
	Elbow[3]	91.51	0.03	90.76	92.17	93.21	92.96
	Wrist[4]	52.19	0.00	59.28	51.14	65.53	64.39
	Hip[5]	99.70	0.00	99.42	99.50	99.59	99.59
FM2[6]	Head[1]	99.94	97.27	100.00	99.96	99.96	99.96
	Shoulder[2]	99.91	98.41	100.00	100.00	99.98	99.98
	Elbow[3]	29.55	95.99	99.45	99.79	98.92	99.81
	Wrist[4]	1.95	17.82	31.73	25.07	31.01	45.33
	Hip[5]	79.95	100.00	100.00	99.94	99.96	99.99
FM3[6]	Head[1]	99.34	22.58	99.26	99.39	99.54	99.56
	Shoulder[2]	99.65	24.17	99.61	99.69	99.68	99.62
	Elbow[3]	76.41	23.07	92.88	94.01	94.59	94.62
	Wrist[4]	44.05	2.85	52.58	47.03	60.14	61.48
	Hip[5]	95.08	23.18	99.56	99.60	99.68	99.69

[1] Head uses head and neck joints. [2] Shoulder uses rshoulder, lshoulder and chest joints.

[3] Elbow uses relbow and lelbow joints. [4] Wrist uses rwrist and lwrist joints. [5] Hip uses rhip, lhip and pelvis joints.

[6] FM# does matching threshold to $26\ pixels$

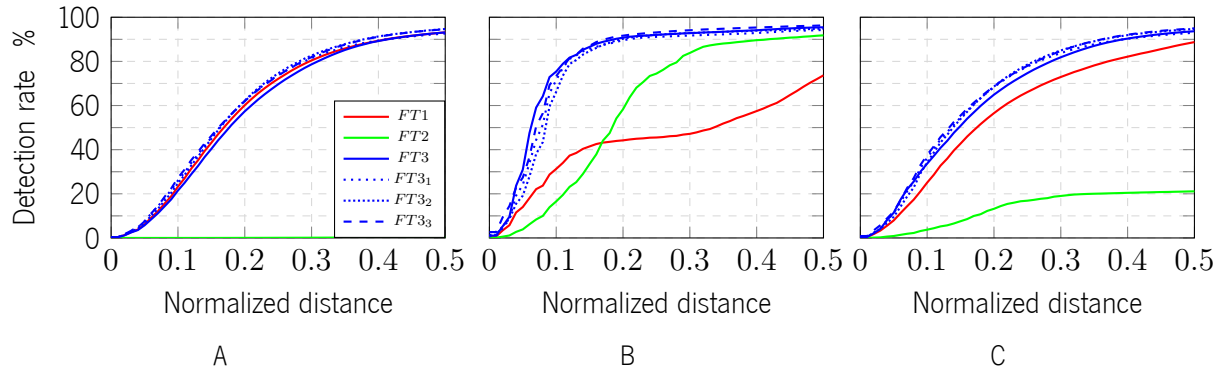


Figure 7.4: PCKh total for all training sets, $\boldsymbol{FT\#}$, in each ToF PoV experimental scenarios, $\boldsymbol{FM\#}$: (A) ToF front row PoV (FM1); (B) ToF driver PoV (FM2); and (C) ToF front row and driver PoV (FM3). Color gradient represents different combinations of real datasets, $\boldsymbol{RD^{\#}_{t'}}$. Dotted and dashed lines represent different combinations of synthetic datasets, $\boldsymbol{SD^{\#}_{t'}}$, added to the the full combination of real datasets, $\boldsymbol{FT3}$.

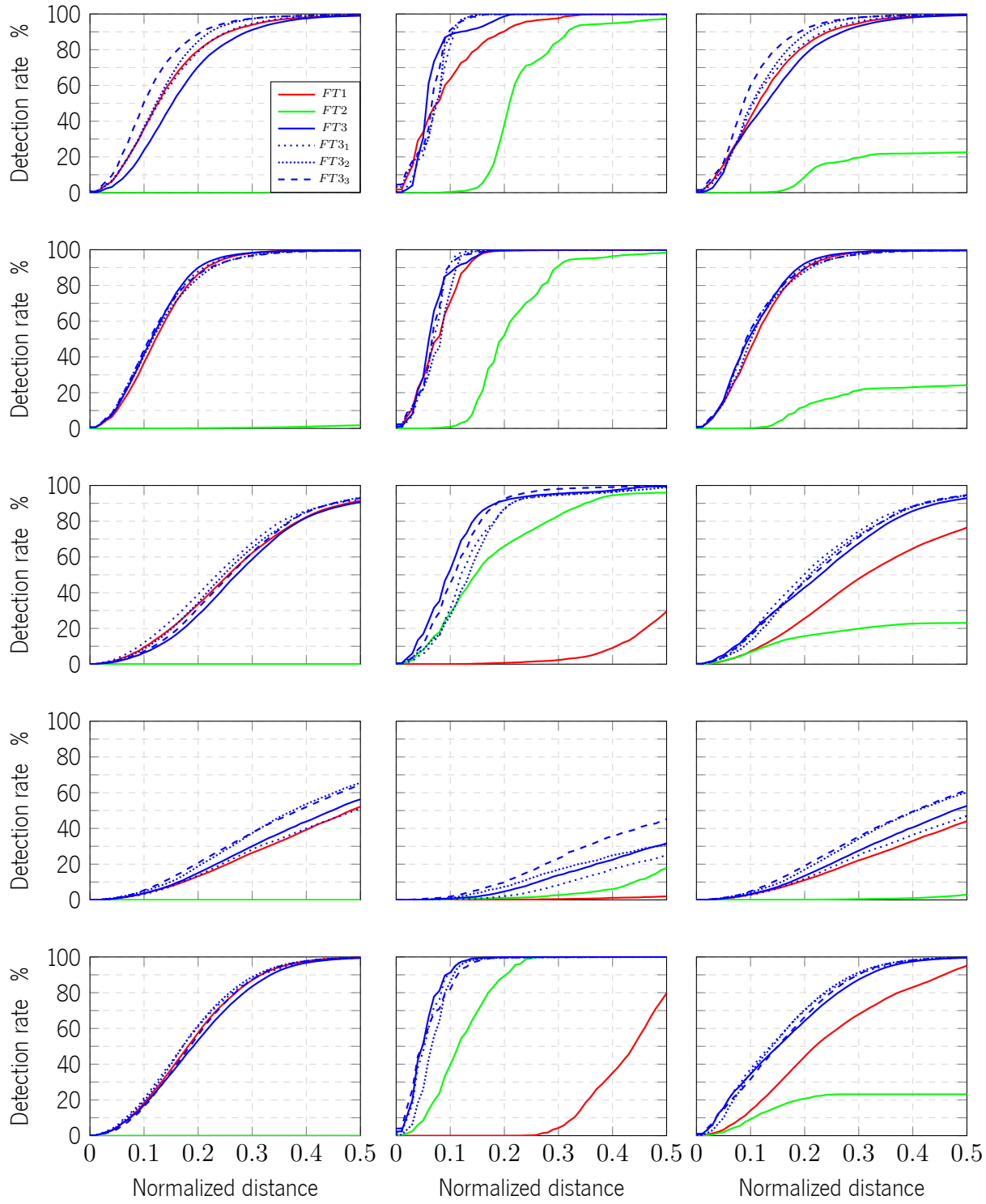


Figure 7.5: PCKh total for all training sets, $\boldsymbol{FT\#}$, in each ToF PoV experimental scenarios, $\boldsymbol{FM\#}$: (1^{st} column) ToF front row PoV (FM1); (2^{nd} column) ToF driver PoV (FM2); and (3^{rd} column) ToF front row and driver PoV (FM3). (1^{st} row) Head; (2^{nd} row) Shoulder; (3^{rd} row) Elbow; (4^{th} row) Wrist; and (5^{th} row) Hip. Color gradient represents different combinations of real datasets, $\boldsymbol{RD^{\#}_{t'}}$. Dotted and dashed lines represent different combinations of synthetic datasets, $\boldsymbol{SD^{\#}_{t'}}$, added to the the full combination of real datasets, $\boldsymbol{FT3}$

7.2.3 On road demonstration

A final evaluation was performed in a real scenario, with the ToF camera placed in a BMW i3 with a driver PoV, serving as a system demonstration in the Frontiers in Human Machine Interfaces – INNOVATIVE CAR EXPERIENCE conference (Appendix C). The system was also disclosed in media services (e.g. University of Minho, Jornal de Noticias, and SIC [Appendix E]), as well as DoCEIS 2019 conference (Appendix D).

7.3 Discussion and Conclusions

In this chapter, a toolchain capable of evaluating (i.e. quantitatively and qualitatively) human body pose detection algorithms was presented. The toolchain is able to interface with algorithms through IC, allowing for a transversal compatibility algorithm implementation frameworks (e.g. caffe, tensorflow, etc). Quantitative evaluation is provided through pre-established metrics, an easy interface with the dataset standard from chapters 4 and 5, and also the access to per-frame inferrence and ground-truth visualization. Qualitative evaluation also helps the user to test the method in a real live scenario, for testing or demonstration purposes. This is possible with the same IC implementation, giving access to ToF real time frames.

In terms of final algorithmic evaluation, Figures 7.4, 7.5 and Tables 7.4, 7.5 demonstrate the feasibility of using the algorithm in an in-car scenario, with reduced restrictions of ToF in-car placement (i.e. driver or front row PoV). To enhance the algorithm's final accuracy and placement generalization, four datasets were used, $\boldsymbol{RD^{\#}_{t'}, SD^{\#}_{t'}}$, from the previously mentioned chapters. When trained with real data only, the results show that training with both PoV scenarios ($\boldsymbol{FT3}$) leads to better results. This understanding comes from the higher PCKh@0.5 total score and Area Under Curve (AUC) achieved in $\boldsymbol{FM3}$, with 93.67% and 63.46% respectively. Finally, the addition of synthetic data to the best real data scenario, $\boldsymbol{FT3}$, also improved the method's accuracy. This is shown in Table 7.4, where $\boldsymbol{FT3_3}$ has PCKh@0.5 total score and AUC of 94.97% and 65.99% respectively, and in more detail in Table 7.5, with wrist joints improving almost 10% in PCKh@0.5 score. Better improvements could be achieved if the real to synthetic dataset ratio was increased (e.g. as it was shown in Chapter 5), considering that the final one was 1:1 ratio. Qualitative results in Figure 7.6 show how the best trained model, $\boldsymbol{FT3_3}$, outperforms the other two trained models (i.e. $\boldsymbol{FT1, FT2}$) in any ToF PoV scenario, achieving better accuracy and ToF placement generalization. Figure 7.7 show the method failing to accuratly estimate the position of the wrist joints, something that was evident with the quantitative results shown in Figure 7.5. This failure is associated with proximity of the joint to the edge of the image, or training error from ground-truth (i.e. proximity of the joint to the ToF sensor, or kinematic forwarding error propagation). Finally, worst case estimation scenarios are shown in Figure 7.8, where the model fails to

estimate the entire full body, due to proximity of body parts to the ToF sensor dead-zone. In conclusion, the algorithm is able to work inside a vehicle, irrespective of the ToF sensor placement (either front row PoV or driver PoV). Overall, it is able to detect 40% of the body joints with a distance error lower than 2.6 pixels, 70% for 5.2 pixels, and >90% for 9.1 pixels.

Figure 7.6: Good quantitative results of the final human body pose detection method for three training configurations, $\boldsymbol{FT\#}$: (1^{st} row) represent $\boldsymbol{FT1}$ in both ToF PoV scenarios; (2^{nd} row) represent $\boldsymbol{FT2}$ in both ToF PoV scenarios; (3^{rd} row) represent $\boldsymbol{FT3_3}$ in both ToF PoV scenarios; (1^{st} to 3^{rd} column) represent $\boldsymbol{FM1}$; (4^{th} to 6^{th} column) represent $\boldsymbol{FM2}$. Good results consider the inferrence from the best training, $\boldsymbol{FT3_3}$, where it is possible to show that it outperforms the other two trainings in any ToF PoV scenario.

Figure 7.7: Joint failure quantitative results of the final human body pose detection method for three training configurations, $\boldsymbol{FT\#}$: (1^{st} row) represent $\boldsymbol{FT1}$ in both ToF PoV scenarios; (2^{nd} row) represent $\boldsymbol{FT2}$ in both ToF PoV scenarios; (3^{rd} row) represent $\boldsymbol{FT3_3}$ in both ToF PoV scenarios; (1^{st} to 3^{rd} column) represent $\boldsymbol{FM1}$; (4^{th} to 6^{th} column) represent $\boldsymbol{FM2}$. Joint failure is associated with proximity of the joint to the edge of the image (i.e. Wrists), or training error from ground-truth (i.e. proximity of the joint to the ToF sensor, or kinematic forwarding error propagation).

Figure 7.8: Full body failure quantitative results of the final human body pose detection method for three training configurations, $\boldsymbol{FT\#}$: (1^{st} row) represent $\boldsymbol{FT1}$ in both ToF PoV scenarios; (2^{nd} row) represent $\boldsymbol{FT2}$ in both ToF PoV scenarios; (3^{rd} row) represent $\boldsymbol{FT3_3}$ in both ToF PoV scenarios; (1^{st} to 3^{rd} column) represent $\boldsymbol{FM1}$; (4^{th} to 6^{th} column) represent $\boldsymbol{FM2}$. Full body failure is associated with missing feature information due to the presence of body parts in the ToF sensor dead-zone.

Chapter 8

Conclusion and Future Work

Contents

In this thesis, an algorithm for in-car human body pose detection was developed with a suitable ground-truth system.

Algorithmic development was focused, in a first stage, on selection and evaluation of state-of-the-art methods, followed by the customization of the selected algorithm. Finally the method was trained and tested with real and synthetic datasets to demonstrate its accuracy and its generalization capabilities with respect to the Time-of-Flight (ToF) sensor placement.

Synthetic datasets were generated through a novel toolchain. A synthetic dataset generated by the toolchain was also made publicly available. The toolchain demonstrated to be computationally efficient (up to 4 generated frames per second), while demonstrating its potential for increased algorithmic accuracy during body pose estimation in an in-car scenario.

Real datasets were similarly generated through a novel toolchain making use of a specific setup consisting in an inertial suit, a global positioning system and a ToF camera, coupled with a set of calibration procedures. A toolchain generated dataset was also made publicly available.

The motion capture system (i.e. inertial suit) was thoroughly evaluated, full body analysis showed highest errors for sensors in segments with increased soft tissue, or performing an higher range of motion. This source of error propagates into the joints' positions analysis, due to its inherited kinematic forwarding pipeline, showing the highest errors in the furthest body joints wrt. to the pelvis joint.

As stated, final algorihmic training and evaluation was performed with two real datasets and two synthetic datasets (i.e. aprox. 180k samples), giving the method enough ToF in-car placement generalization and inferrence accuracy. Algorithm final accuracy in Chapter 7 did not achieve better results then in Chapter 6 for the wrist joints (i.e. for the driver ToF Point-of-View (PoV) scenario, [Figures 6.23 vs. 7.5:FM2]). Although evaluation sets being different, it can also be related to the quality of wrist ground-truth in each datasets. In Chapter 6 wrist joints are manually labeled, contrary with the real dataset in Chapter 7 that suffers from kinematic forwarding propagation errors. It is well understood that dataset generation can be bruteforced with manual labeling, however the required human/time resources increase substantialy with ground-truth complexity. This conclusion does not remove the value of the datasets generated or that can be generated by the proposed toolchains (real and synthetic).

8.1 Contributions

In the following list, the contributions of this thesis are enumerated:

- Ground-truth Inertial and Magnetic Measurement Units (IMMUs) motion capture system evaluation:

  - A method for static evaluation of IMMUs sensors.
  - A method for static full body evaluation of motion capture systems.
  - A toolchain for dynamic sensor evaluation of IMMUs motion capture systems. Sensor mean error estimated in 5^o across all motion planes.
  - A toolchain for dynamic full body evaluation of IMMUs motion capture systems. Joint mean error estimated in $30\ mm$ for the worst case.
- A toolchain for the generation of real in-car datasets:
  - Implementation of ToF automotive standardized products.
  - Methods for spatial and temporal calibration between ToF, optical (Vicon) and inertial (MVN Awinda) systems.
  - GPRD compliant dataset recording.
  - Evaluation of human body pose detection for in-car scenario with different datasets;
  - Two toolchain generated datasets were made publicly available: MoLa R8.7k InCar Dataset [102], and MoLa R10k InCar Dataset [108].
- A toolchain for the generation of realistic synthetic in-car datasets:
  - In-Car 3D scenario, human model and motion profile, and ToF sensor customization.
  - Realistic anthropometric motion constraints for synthetic human models.
  - Method for 3D scene data validation, through human motion collision detection.
  - Evaluation of human body pose detection scenarios with different real and synthetic dataset mixing ratios, showing accuracy improvements with the addition of synthetic datasets.
  - Evaluation of different feature types (i.e. depth, point cloud, ground-truth, body part segmentation) for human body pose detection, showing that point cloud feature improve inferrence accuracy comparatively to 2D depth features;
  - A publically available dataset [109], generated by the proposed toolchain.
- A human body pose detection algorithm for in-car scenario:
  - Selection and evaluation of state-of-the-art body pose detection methods with public depth based dataset.
  - Computational customization of state-of-the-art body pose detection method.

  - Novel data augmentation method for state-of-the-art body pose detection method training.
  - Network customization (i.e. 3^{rd} branch) for joint visibility detection.
- A toolchain for algorithmic online/offline evaluation.
- Final evaluation of the human body pose detection algorithm with a mix of real and synthetic datasets, generated by the proposed toolchains.

8.2 Future Work

The list of envisioned future work targets the outputs of multiple chapters.

Chapter 4 still presents several limitations in regard to the ground-truth system, requiring improvements in three main aspects: (1) Inertial based system correction of kinematic forward based errors on joints such as wrists or hips; (2) Inertial full body calibration error, creating a systematic relative joint error; and (3) Vicon to Awinda spatial alignment is crutial for a good projection of all inertial based relative joints into an absolute positioning. The first aspects can be improved with base improvements from the manufacturer or with different products (not restricted to inertial ones). A second improvement can be achieved with the creation of an extra toolchain for manual labeling that allows the user to fine-tune specific problematic joints (e.g. wrists and hips). The second aspect also requires improvements on the suit itself, however an extra toolchain can also be developed to allow for an initial full body calibration, where the relative full body joint error can be compensated. It is imperative that this toolchain does not add substantial setup time to the dataset recording procedure. For the third and final aspect, better fixation methods can be used for the head inertial sensor to decrease the orientation error between sensor and Vicon head tracker. Also, it is possible to change the Vicon head tracker system for a mobile one (i.e. SMARTTRACK [120]), considerably enhancing the possibility for dataset generalization through in-car on-road recording.

Chapter 5 still does not achieve the most realistic human motion profile, relying only on brute-force Incremental or Gaussian motion profiles. The ability to synthetically recreate human behaviour would be another important feature, enlarging the applicability of our dataset towards other monitoring tasks (like action recognition). Hereto, fusion of real human motion capture data with synthetic scenarios could be employed. However, issues such as collision detection between animated models and synthetic car models would have to be handled. Besides the currently supported pose and segmentation maps, another relevant output to be added would be the gaze for each human model. Another important aspect is scene realism, as discriminative algorithms seem to improve their accuracy proportionally to the training data realism. In this regard, future work may focus on improving the ToF noise characterization or the used Neural Style Transfer

(NST) methods, as well as the RGB image rendering. Specifically to the NST rendering, it is also important to improve its computational performance, that still limits the data throughput of the toolchain.

Chapter 6 presents a customized state-of-the-art algorithm for 2D human body pose estimation. This inferring label format can be a limitation to specific in-car use-cases currently considered in Bosch roadmap. A future change to be considered would be the customization for 3D human body pose estimation. This change would have little to no problem in training due to the already present 3D ground-truth data in the generated datasets (real and synthetic).

Regarding the future developments related with the output from this thesis, there are already two new project ideas that make use of the knowledge and hardware/software infrastructure generated during this thesis. These new project ideas were already submitted as R&D solutions in a new future phase for Bosch and UMinho partnership. The new project ideas are:

- Violent Action Recognition and Left Items Detection with Categorization – the purpose of this project is to develop an algorithm for violent action recognition, making use of the human body pose detection algorithm and the development pipeline created in this thesis. This project intends to expand the current project towards a distinct field in computer vision, i.e. object detection and categorization.
- Monitoring stain/dirt and damage in the scope of Shared Autonomous Vehicles (SAV) – The purpose of this project is to use the current knowledge in computer vision, hardware (sections 2.1.2, 2.1.4 and 2.1.5), software (Section 2.2.1) and expand it into the field of automated interior monitoring inspection, specifically stain/dirt and damage detection.

Considering the fact that the two mentioned iterations are final use-cases, both of them will consider hardware/software embedded deployment, with automotive standards contextualization. In this new project ideas, it is expected that the developed real dataset toolchain may be updated, or may even evolve, especially if it is of intention, in this regard, to iterate into a portable toolchain capable of recording while driving on a road. The same future update scenario may occur in what concerns the synthetic dataset toolchain. If there is a need to improve the realism of the output data, or even add more information to it (i.e. temporal movement), then there should and will be improvements applied to it in future projects. Regarding the Bosch business perspective, this thesis work serves as tools (ground-truth and body pose detection algorithm) for the R&D focused on use-cases related with SAV. With future SAE 5 autonomous driving, the need for SAV will increase, and with it comes the car interior inspection and human safety.

Bibliography

[1] J. A. e. Sousa, "LREC - Metrologia industrial," 2010. xi, 3

[2] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," Conference on Computer Vision and Pattern Recognition 2011, pp. 1297–1304, 2011. xi, 6, 7, 13, 20, 105

[3] K. Buys, C. Cagniart, A. Baksheev, T. De Laet, J. De Schutter, and C. Pantofaru, "An adaptable system for RGB-D based human body detection and pose estimation," Journal of Visual Communication and Image Representation, vol. 25, no. 1, pp. 39–52, 2014. xi, 6, 7, 8

[4] M. H. Tsai, K. H. Chen, and I. C. Lin, "Real-time upper body pose estimation from depth images," Proceedings - International Conference on Image Processing, ICIP, vol. 2015-Decem, pp. 2234–2238, 2015. xi, 6, 7, 8, 9

[5] S. Knoop, S. Vacek, and R. Dillmann, "Sensor fusion for 3D human body tracking with an articulated 3D body model," International Conference on Robotics and Automation, no. May, pp. 1686–1691, 2006. xi, 6, 10, 11

[6] T. Xing, Y. Yu, Y. Zhou, and S. Du, "Markerless Motion Capture of Human Body Using PSO with Single Depth Camera," 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission, pp. 192–197, 2012. xi, 6, 10, 11, 12

[7] M. Ye, Y. Shen, C. Du, Z. Pan, and R. Yang, "Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 8, pp. 1517–1532, 2016. xi, 6, 10, 12, 13

[8] A. Baak, M. Müller, G. Bharaj, H.-P. Seidel, and C. Theobalt, "A Data-Driven Approach for Real-Time Full Body Pose Reconstruction from a Depth Camera," Consumer Depth Cameras for Computer Vision, pp. 71–98, 2013. xi, 13, 14

[9] Xsens®, MVN quick setup sheet. Xsens Technologies B.V., 2015. xi, 25, 26, 27

[10] Vicon®, Plug-in-Gait Marker Placement. 2006. xii, 30

[11] Vicon®, Plug-in Gait Reference Guide. 2017. xii, 30, 31

[12] M. Poulin, "LEDs and sensing technology enable new ranging applications @ www.ledsmagazine.com," 2014. xii, 33, 34

[13] Microsoft, "kinect v2 @ developer.microsoft.com," 2016. xii, 35, 36

[14] T. Instruments, "OPT8241-CDK-EVM OPT8241 3D Time-of-Flight (ToF) Sensor Evaluation Module | TI.com," 2018. xii, 36

[15] Sick, "V3S110-1AABAAB | Vision SICK," 2018. xii, 36

[16] Melexis, "EVK75023 @ www.melexis.com," 2016. xii, 38

[17] Melexis, Time-of-flight basics. Melexis, 2017. xii, 35, 41

[18] Nvidia, "Tensor Cores in NVIDIA Volta Architecture | NVIDIA," 2018. xii, 43

[19] Xsens®, "MVN User Manual, Body Suit 1," 2013. xiii, 25, 27, 51, 63

[20] Melexis, Software User Manual EVK75123. Melexis, 2016. xiv, 75

[21] P. F. Felzenszwalb, R. B. Girshick, D. Mcallester, and D. Ramanan, "Object Detection with Discriminatively Trained Part Based Models," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, pp. 1–20, sep 2009. xvii, 135, 136, 137

[22] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake, "Efficient human pose estimation from single depth images," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, pp. 2821–2840, dec 2013. xviii, 6, 7, 137, 138, 139

[23] H. Y. Jung, S. Lee, Y. S. Heo, and I. D. Yun, "Random tree walk toward instantaneous 3D human pose estimation," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 07-12-June, pp. 2467–2474, 2015. xviii, 135, 139, 140

[24] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 6517–6525, 2017. xviii, 135, 140

[25] Z. Cao, T. Simon, S. E. Wei, and Y. Sheikh, "Realtime multi-person 2D pose estimation using part affinity fields," Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 1302–1310, nov 2017. xviii, xix, 21, 96, 125, 135, 141, 142, 150, 151

[26] LISA, "cvrr.ucsd.edu." 1

[27] N. Das, E. Ohn-Bar, and M. M. Trivedi, "On Performance Evaluation of Driver Hand Detection Algorithms: Challenges, Dataset, and Metrics," IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC, vol. 2015-Octob, pp. 2953–2958, 2015. 1

[28] E. Ohn-Bar, S. Martin, A. Tawari, and M. M. Trivedi, "Head, eye, and hand patterns for driver activity recognition," Proceedings - International Conference on Pattern Recognition, pp. 660–665, 2014. 1

[29] E. Ohn-Bar and M. M. Trivedi, "Hand gesture recognition in real time for automotive interfaces: A multimodal vision-based approach and evaluations," IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 6, pp. 2368–2377, 2014. 1, 13, 14

[30] E. Ohn-Bar and M. M. Trivedi, "Fast and Robust Object Detection Using Visual Subcategories," IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 179–184, 2014. 1

[31] S. University, "SAIL @ aicenter.stanford.edu." 1

[32] M. Schwager, C. Gerdes, M. Kochenderfer, and M. Pavone, "Safe Feedback Interactions in Human-Autonomous Vehicle Systems @ aicenter.stanford.edu.". 1

[33] W. Ju, "Understanding Driver State in Laboratory and Naturalistic Environments @ aicenter.stanford.edu.". 1

[34] M. Agrawala, M. Bernstein, and J. Landay, "Human Behaviors and Interaction for In-Car Experiences @ aicenter.stanford.edu.". 1

[35] I. Lillo, J. C. Niebles, and A. Soto, "A Hierarchical Pose-Based Approach to Complex Action Understanding Using Dictionaries of Actionlets and Motion Poselets," Cvpr, pp. 1981–1990, 2016. 1

[36] J. Brauer and M. Arens, "Reconstructing The Missing Dimension: From 2D To 3D Human Pose Estimation," in REACTS - REcognition and ACTion for Scene Understanding (CAIP workshop), no. September, pp. 25–39, 2011. 1

[37] J. Brauer, H&quot, W. ubner, and M. Arens, "Generative 2D and 3D human pose estimation with vote distributions," Advances in Visual Computing, pp. 470–481, 2012. 1, 15

[38] D. Muench, W. Huebner, M. Arens, and F. Iosb, "Generalized Hough transform based time invariant action recognition with 3D pose information," vol. 9253, no. c, pp. 1–11, 2014. 1, 13, 14

[39] N. Hesse, G. Stachowiak, T. Breuer, and M. Arens, "Estimating Body Pose of Infants in Depth Images Using Random Ferns," Proceedings of the IEEE International Conference on Computer Vision, vol. 2016-Febru, pp. 427–435, 2016. 1, 6, 7, 9

[40] J. Valmadre and S. Lucey, "Deterministic 3D Human Pose Estimation Using Rigid Structure," Structure, vol. 6313, pp. 467–480, 2010. 1

[41] A. Saran, D. Teney, and K. M. Kitani, "Hand parsing for fine-grained recognition of human grasps in monocular images," IEEE International Conference on Intelligent Robots and Systems, vol. 2015-Decem, pp. 5052–5058, 2015. 1

[42] H. Alismail, Direct Pose Estimation and Refinement. PhD thesis, 2016. 1

[43] L. Shen and P. R. Stopher, "Using SenseCam to pursue "ground truth" for global positioning system travel surveys," Transportation Research Part C: Emerging Technologies, vol. 42, pp. 76–81, 2014. 3

[44] IPQ, "IPQ-Metrologia @ www1.ipq.pt," 2016. 3

[45] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," Computer Vision and Image Understanding, vol. 104, no. 2-3 SPEC. ISS., pp. 90–126, 2006. 4

[46] K. Mitobe, T. Kaiga, T. Yukawa, T. Miura, H. Tamamoto, A. Rodgers, and N. Yoshimura, "Development of a motion capture system for a hand using a magnetic three dimensional position sensor," ACM SIGGRAPH 2006 Research posters on - SIGGRAPH '06, p. 102, 2006. 5

[47] J. R. Mcneal, P. D, H. A. S. Eastern, and P. Education, "The United States Olympic Committee uses the Polhemus LIBERTY ™ To Research the Effects of Acute Static Stretch on Joint Position Sense in the Shoulder," Computer, vol. 4777, pp. 800–802, 2003. 5

[48] S. J. Lee, Y. Motai, and H. Choi, "Tracking Human Motion With Multichannel Interacting Multiple Model," Industrial Informatics, IEEE Transactions on, vol. 9, no. 3, pp. 1751–1763, 2013. 5

[49] L. Sigal, A. O. Balan, and M. J. Black, "HumanEva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion," International Journal of Computer Vision, vol. 87, no. 1-2, pp. 4–27, 2010. 5, 15, 100, 130

[50] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," Studies in Computational Intelligence, vol. 411, pp. 119–135, 2013. 5, 16, 20, 105

[51] CMU, "CMU Dataset @ mocap.cs.cmu.edu," 2016. 5, 15, 100, 130

[52] S. Rahmatalla, T. Xia, M. Contratto, G. Kopp, D. Wilder, L. Frey Law, and J. Ankrum, "Three-dimensional motion capture protocol for seated operator in whole body vibration," International Journal of Industrial Ergonomics, vol. 38, no. 5-6, pp. 425–433, 2008. 5

[53] G. Wu, S. Siegler, P. Allard, C. Kirtley, A. Leardini, D. Rosenbaum, M. Whittle, D. D. D'Lima, L. Cristofolini, H. Witte, O. Schmid, and I. Stokes, "ISB recommendation on definitions of joint coordinate system of various joints for the reporting of human joint motion—part I: ankle, hip, and spine," Journal of Biomechanics, vol. 35, no. 4, pp. 543–548, 2002. 5, 25

[54] D. Roetenberg, H. Luinge, and P. Slycke, "Xsens MVN: Full 6DOF Human Motion Tracking Using Inertial Sensors," tech. rep., Xsens Technologies, 2013. 5, 25

[55] M. Orozco, Assessment of Postural Deviations Associated Errors in the Analysis of Kinematics Using Inertial and Magnetic Sensors and a Correction Technique Proposal by Assessment of Postural Deviations Associated Errors in the Analysis of Kinematics Using Inertial a. PhD thesis, University of Toronto, 2015. 5, 68, 101

[56] Y. Pekelny and C. Gotsman, "Articulated object reconstruction and markerless motion capture from depth video," Computer Graphics Forum, vol. 27, no. 2, pp. 399–408, 2008. 6, 10, 11

[57] D. Demirdjian and C. Varri, "Driver pose estimation with 3D Time-of-Flight sensor," in 2009 IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems, pp. 16–22, IEEE, mar 2009. 6

[58] M. Ye, X. Wang, R. Yang, L. Ren, and M. Pollefeys, "Accurate 3D pose estimation from a single depth image," IEEE International Conference on Computer Vision, pp. 731–738, nov 2011. 6

[59] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, "Real-Time Human Pose Tracking from Range Data," European Conference on Computer Vision, pp. 738–751, 2012. 6, 10, 12, 13

[60] Z. Li, Single View Human Pose Tracking. PhD thesis, 2013. 6, 10, 12

[61] M. Sigalas, M. Pateraki, and P. Trahanias, "Full-Body Pose Tracking - The Top View Reprojection Approach," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, pp. 1569–1582, aug 2016. 6

[62] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, "Towards Viewpoint Invariant 3D Human Pose Estimation," 2016. 6, 95, 143

[63] B. Crabbe, A. Paiement, S. Hannuna, and M. Mirmehdi, "Skeleton-Free Body Pose Estimation from Depth Images for Movement Analysis," in Proceedings of the IEEE International Conference on Computer Vision, vol. 2016-Febru, pp. 312–320, 2016. 6, 7, 9, 15

[64] C. Plagemann, V. Ganapathi, D. Koller, and S. Thrun, "Real-time identification and localization of body parts from depth images," Proceedings - IEEE International Conference on Robotics and Automation, pp. 3108–3113, 2010. 7

[65] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, "Real time motion capture using a single time-of-flight camera," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR2010), pp. 755–762, 2010. 13

[66] V. Stohne, Real-time filtering for human pose estimation using multiple Kinects. PhD thesis, 2014. 13, 14

[67] G. Borghi, M. Venturelli, R. Vezzani, and R. Cucchiara, "POSEidon: Face-from-Depth for driver pose estimation," Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 5494–5503, 2017. 15, 100

[68] S. Martin, K. Yuen, and M. M. Trivedi, "Vision for Intelligent Vehicles & Applications (VIVA): Face detection and head pose challenge," IEEE Intelligent Vehicles Symposium, Proceedings, vol. 2016-Augus, no. Iv, pp. 1010–1014, 2016. 15

[69] S. E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, Convolutional pose machines. PhD thesis, 2016. 15

[70] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid, "Learning from synthetic humans," Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 4627–4635, 2017. 16, 130

[71] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 7, pp. 1325–1339, 2014. 16

[72] M. Loper, N. Mahmood, J. Romero, G. Pons-moll, and M. J. Black, "SMPL : A Skinned Multi-Person Linear Model," ACM Trans. Graphics (Proc. SIGGRAPH Asia), vol. 34, no. 6, pp. 248:1—248:16, 2015. 16, 17

[73] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), no. 600388, pp. 3234–3243, 2016. 16, 20, 130

[74] F. M. Carlucci, P. Russo, and B. Caputo, "A deep representation for depth images from synthetic data," Proceedings - IEEE International Conference on Robotics and Automation, pp. 1362–1369, 2017. 16, 17, 130

[75] S. Pini, F. Grazioli, G. Borghi, R. Vezzani, and R. Cucchiara, "Learning to generate facial depth maps," Proceedings - 2018 International Conference on 3D Vision, 3DV 2018, pp. 634–642, 2018. 17

[76] X. Liu, W. Liang, Y. Wang, S. Li, and M. Pei, "3D head pose estimation with convolutional neural network trained on synthetic images," Proceedings - International Conference on Image Processing, ICIP, vol. 2016-Augus, pp. 1289–1293, 2016. 17

[77] B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, O. Lehmann, T. Chen, A. Hutter, S. Zakharov, H. Kosch, and J. Ernst, "DepthSynth: Real-Time Realistic Synthetic Data Generation from CAD Models for 2.5D Recognition," Proceedings - 2017 International Conference on 3D Vision, 3DV 2017, pp. 1–10, feb 2018. 17

[78] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree, "BlenSor: Blender sensor simulation toolbox," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6939 LNCS, no. PART 2, pp. 199–208, 2011. 17

[79] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3D reconstruction and tracking," Proceedings - 2nd Joint 3DIM/3DPVT Conference: 3D Imaging, Modeling, Processing, Visualization and Transmission, 3DIMPVT 2012, pp. 524–530, 2012. 17

[80] T. Iversen and D. Kraft, "Generation of synthetic Kinect depth images based on empirical noise model," Electronics Letters, vol. 53, no. 13, 2017. 17, 110

[81] J. Borges, B. Oliveira, H. Torres, N. Rodrigues, S. Queirós, M. Shiller, V. Coelho, J. Pallauf, J. H. Brito, J. Mendes, and J. C. Fonseca, "Automated generation of synthetic in-car dataset for human body pose detection," in VISIGRAPP 2020 - Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, vol. 5, pp. 550–557, SCITEPRESS - Science and Technology Publications, 2020. 23, 102

[82] H. R. Torres, B. Oliveira, J. Fonseca, S. Queirós, J. Borges, N. Rodrigues, V. Coelho, J. Pallauf, J. Brito, and J. Mendes, "Real-Time Human Body Pose Estimation for In-Car Depth Images," in IFIP Advances in Information and Communication Technology, vol. 553, pp. 169–182, Springer, Cham, may 2019. 23, 133

[83] X. T. B.V., MVN Studio real-time network streaming. No. March, Xsens®, 2015. 25, 27, 29

[84] M. Kok, J. D. Hol, and T. B. Schön, "An optimization-based approach to human body motion capture using inertial sensors," IFAC Proceedings Volumes (IFAC-PapersOnline), vol. 19, pp. 79–85, 2014. 26

[85] Xsens®, "MVN Studio hardware synchronization," 2015. 29

[86] M. L. Systems, C3D File Format User Guide. 29, 33

[87] Vicon®, Vicon DataStream SDK Developer ' s Manual. No. January, 2013. 30, 31

[88] Vicon®, Vicon Nexus Reference Guide Contents. 2010. 31

[89] Vicon®, Vicon Lock+. 33

[90] W. Yuan, R. E. Howard, K. J. Dana, R. Raskar, A. Ashok, M. Gruteser, and N. Mandayam, "Phase messaging method for time-of-flight cameras," 2014 IEEE International Conference on Computational Photography, ICCP 2014, 2014. 34

[91] T. Breuer, C. Bodensteiner, and M. Arens, "Low-cost commodity depth sensor comparison and accuracy analysis," SPIE Security + Defence, vol. 9250, p. 92500G, 2014. 35

[92] Q. Wei, J. Shan, H. Cheng, Z. Yu, L. Bai, and H. Zhao, "A method of 3D human-motion capture and reconstruction based on depth information," 2016 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2016, pp. 187–192, 2016. 36

[93] Gener8, "royale release note v3.19 - pmd ToF pico family software," 2018. 41, 77

[94] Elektrobit, "EB Assist ADTF - Elektrobit," 2018. 44, 53

[95] Blender, "docs.blender.org." 47

[96] Nanotec, "SMCI47-S-2 - Closed-Loop Stepper Motor... |... | NANOTEC," 2018. 51

[97] Nanotec, "COM Interface (VB, C#, C++) | NANOTEC," 2018. 51

[98] D.-J. Kroon, Segmentation of the Mandibular Canal in Cone-Beam CT Data. PhD thesis, University of Twente, Enschede, The Netherlands, dec 2011. 61, 88

[99] Z. Zhang, "A flexible new technique for camera calibration," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000. 85

[100] L. E. Kavraki, Protein-Ligand Docking, Including Flexible Receptor-Flexible Ligand Docking. Openstax CNX, 2007. 85

[101] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis," 2016. 95

[102] J. Borges, S. Queirós, B. Oliveira, H. Torres, N. Rodrigues, V. Coelho, J. Pallauf, J. Henrique Brito, J. Mendes, and J. C. Fonseca, "MoLa R8.7k InCar Dataset," 2019. 95, 167, 178

[103] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2D human pose estimation: New benchmark and state of the art analysis," in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2014. 96, 125, 160

[104] J. Martinez, R. Hossain, J. Romero, and J. J. Little, "A Simple Yet Effective Baseline for 3d Human Pose Estimation," in Proceedings of the IEEE International Conference on Computer Vision, 2017. 97, 126

[105] M. Bastioni, "Makehuman, Open Source tool for making 3D characters," 2001. 104

[106] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman, "Controlling perceptual factors in neural style transfer," in Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 3730–3738, 2017. 104, 111, 119, 120

[107] National Aeronautics and Space Administration, "Anthropometry and Biomechanics," Man-Systems Integration Standards NASA-STD-3000, vol. 1, no. 1, pp. 16–45, 2000. 107

[108] J. Borges, S. Queirós, B. Oliveira, H. Torres, N. Rodrigues, V. Coelho, J. Pallauf, J. Henrique Brito, J. Mendes, and J. C. Fonseca, "MoLa R10k InCar Dataset," mar 2019. 123, 167, 178

[109] J. Borges, B. Oliveira, H. Torres, N. Rodrigues, S. Queirós, M. Shiller, V. Coelho, J. Pallauf, J. Henrique Brito, J. Mendes, and J. C. Fonseca, "MoLa S25k InCar Dataset," 2019. 124, 167, 178

[110] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9351, pp. 234–241, 2015. 126

[111] Y. Yang, "Articulated pose estimation with flexible mixtures-of-parts resenting shape," Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, 2011. 136

[112] A. Criminisi, "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning," Foundations and Trends® in Computer Graphics and Vision, vol. 7, no. 2-3, pp. 81–227, 2012. 137

[113] D.-J. Kroon, "Region Growing - File Exchange - MATLAB Central," 2008. 144

[114] Mathworks, "Label connected components in 2-D binary image," 2006. 144

[115] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," jun 2014. 150, 151

[116] D. Osokin, "Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose," pp. 1–5, 2018. 150

[117] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014. 150

[118] S. Ruder, "An overview of gradient descent optimization algorithms," sep 2016. 158

[119] Microsoft, "Interprocess Communications - Windows applications | Microsoft Docs." 166

[120] ART Advanced Realtime Tracking, “Products - ART Advanced Realtime Tracking.” 179

Appendix A

GPRD compliant dataset recording

Universidade do Minho

Informação relativa ao tratamento de dados pessoais no âmbito de projeto investigação

Nome do projeto: **Motion Lab**
Investigador Responsável: João Borges
Centro de Investigação: ALGORITMI
Escola: Escola de Engenharia
Contacto: motionlab@algoritmi.uminho.pt
Contacto do Encarregado de Proteção de dados da UMinho:
http://www.uminho.pt/protecaodados

Apresentação do projeto:
O projeto Motion Lab estuda a postura e movimentos das pessoas dentro do habitáculo do automóvel com vista a desenvolver soluções para carros autónomos, nomeadamente na segurança de passageiros e na interação dos mesmos com o sistema de infoentretenimento. Para obter os dados de que precisa, o projeto convida pessoas a realizar um conjunto de ações dentro de um espaço que reproduz o habitáculo de um automóvel, equipado com vários sistemas de captura que registarão informações de postura e movimento da pessoa em estudo.
Os sistemas de captura registam posturas 2D e 3D, e informação visual em formato de amplitude infravermelha, *depth* e *point-cloud* (Figura 1).

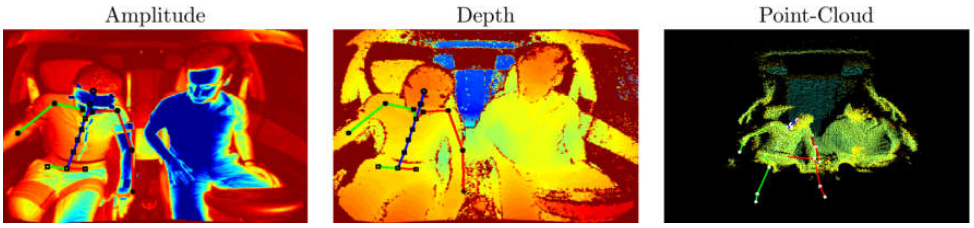


Figura 1 - Informação associada aos dados: Postura 2D e 3D projetadas em imagens de Amplitude, Depth e Point-Cloud.

Categorias de dados pessoais recolhidas:
- Nome;
- Email;
- Posição das articulações do corpo, nomeadamente a postura 2D e 3D projetadas nas imagens;
- Imagens em formato de amplitude infravermelha, *depth* e *point-cloud*

Finalidades do tratamento:
- Constituição de um repositório com informação de posturas e movimentos dentro do habitáculo de automóvel, para investigação científica pela UMinho, e para partilha gratuita com outras comunidades científicas mundiais.
- Publicação de vídeos promocionais do repositório em plataformas como o Youtube. As imagens serão editadas de forma a proteger a identidade dos titulares dos dados.

Licitude para o tratamento:
Consentimento do titular - RGPD Artigo 6º, alínea a): "O titular dos dados tiver dado o seu consentimento para o tratamento dos seus dados pessoais para uma ou mais finalidades específicas;"

Destinatários dos dados:
- O Responsável pelo tratamento manterá um registo de todos os destinatários que tiverem acesso aos dados;

Direitos dos titulares dos dados:

- Direito a solicitar, de forma gratuita, o acesso aos seus dados;
- Direito a retirar o consentimento dado junto do Responsável pelo tratamento, e nessa circunstância cessam todos os tratamentos sobre os seus dados.
- Direito a solicitar a eliminação dos seus dados, a que o Responsável corresponderá num prazo máximo de 120 dias a contar da data do pedido.

Os dados produzidos pela análise dos dados recolhidos, que não identifiquem ou sejam identificáveis com o seu titular estão fora do âmbito do Regulamento Geral sobre a Proteção de Dados, e fora do âmbito deste documento.

☐ **Declaro concordar com a utilização dos meus dados pessoais nas condições apresentadas neste documento.**

Assinatura:
_____
Nome:
_____
Email:
_____

Data: __/__/____

Appendix B

Real dataset toolchain setup

Figure B.1: Real dataset toolchain: hardware setup.

Figure B.2: Real dataset toolchain: car testbed.

Figure B.3: Real dataset toolchain: recording preparation.

Figure B.4: Real dataset toolchain: subject preparation.

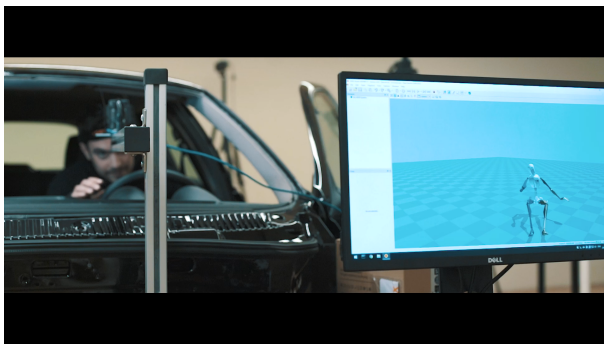


Figure B.5: Real dataset toolchain: recording dataset.

Appendix C

Frontiers in Human Machine Interfaces – INNOVATIVE CAR EXPERIENCE

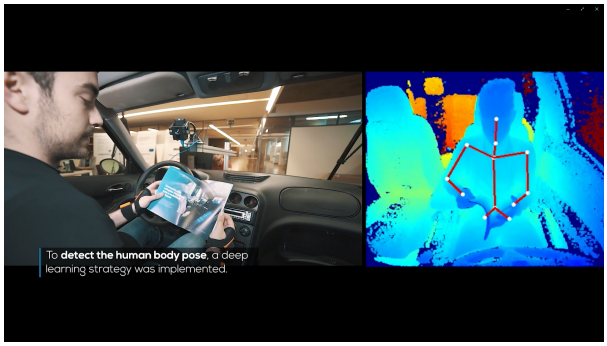


Figure C.1: Innovative car experience: in-car demonstration with subject reading book.

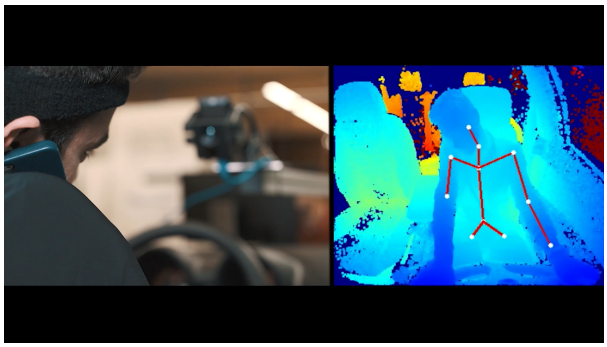


Figure C.2: Innovative car experience: in-car demonstration with subject making a phone call.

Figure C.3: Innovative car experience: oral presentation.

Figure C.4: Innovative car experience: oral demonstration.

Figure C.5: Innovative car experience: live demonstration.

Appendix D

DoCEIS 2019

Figure D.1: DoCEIS19: oral presentation.

Appendix E

System Demonstration

Figure E.1: Media system demonstration: Jornal de Notícias.

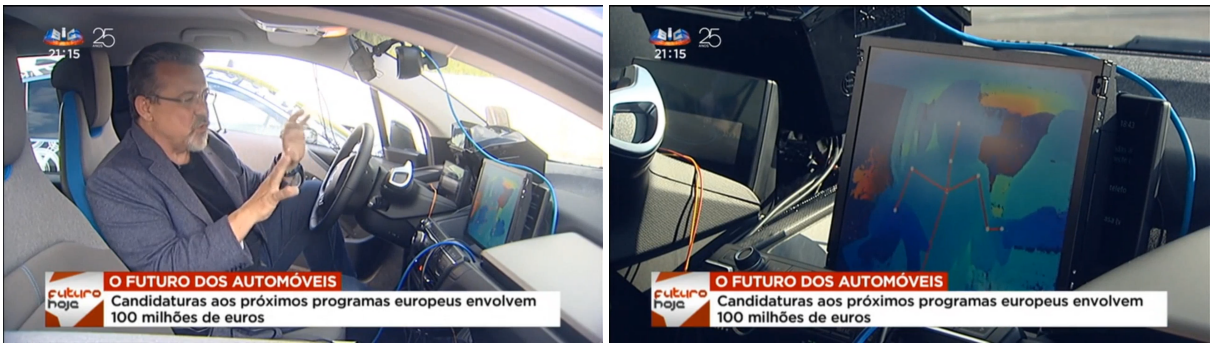


Figure E.2: Media system demonstration: SIC.