Artificial Intelligence Framework for Human-Robot Cooperative Operation through Hand Gesture Recognition

Luis Alejandro Grajeda Blandon

Thesis to obtain the Master of Science degree in

Electrical and Computer Engineering

Supervisors:
Prof. David Alexandre Cabecinhas
Prof. Ricard Marxer

Examination Committee

November 2023

Declaration

Acknowledgements

Firstly, I would like to express my gratitude towards my family and my friends, their support throughout this journey is deeply appreciated and has been key during these past two years.

I would also like to thank my supervisors, Prof. David Alexandre Cabecinhas from *Instituto Superior Técnico* and Prof. Ricard Marxer from *Université de Toulon*, for it is only through their guidance, support, and expertise, that this work has been possible.

Lastly, I also wish to appreciate the support I received from the European Regional Development Fund (FEDER), which funded my research through the Lisbon Regional Operational Program 2020 (LISBOA-01-0145-FEDER-022157), within the scope of the project EMSO-PT-PINFRA/22157/2016_LISBOA-01-01.

Abstract

Underwater human-robot communication and cooperation is a key step towards reducing the burden on divers in a broad range of undersea applications. Since commonly used radio frequency based methods are unavailable in sub-aquatic environments, novel approaches need to be developed and tested. Therefore, this research work proposes to explore the boundaries on vision-based communication with a robot beneath the water surface. Thus, under the scope of this research project, a framework for human-robot interaction has been developed by integrating state-of-the-art neural network models.

The aforementioned framework is based on StaDNet, a model for static and dynamic gesture recognition. We propose a human pose-based approach that allows the extraction and exploitation of spatial and temporal key-points to perform gesture inference. Our human pose estimation method relies on ViT-Pose, a Vision Transformer based model with state-of-the-art performance in both accuracy and speed. Excellent results have been achieved when testing the framework on the ChaLearn LSSII Continuous Gestures Dataset, on which we achieved 81% accuracy during pre-training on a sample of 47 gestures, and on our own in-house underwater dynamic gestures dataset, on which the model achieves an overall 74.4% test accuracy. This framework was integrated with a real vehicle, enabling a diver to communicate with it by performing gestures.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence.

ANN Artificial Neural Network.

ASL American Sign Language.

AUV Autonomous Underwater Vehicle.

CNN Convolutional Neural Network.

CTC Connectionist Temporal Classification.

CVPR Computer Vision and Pattern Recognition.

DG-STA Dynamic Graph-Based Spatial-Temporal Attention.

DSOR Dynamical Systems and Ocean Robotics Lab.

FAROL Free Autonomous Robots for Observations and Labelling.

FMP Feature Map Processor.

FoH Focus on Hands.

FSM Finite State Machine.

GAP Global Average Pooling.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

HRC Human-to-Robot Communication.

ISR Institute for Systems Robotics.

LSSI Large Scale Signer Independant.

LSTM Long Short-Term Memory.

MLP Multi-Layer Perceptron.

RHC Robot-to-Human Communication.

RNN Recurrent Neural Network.

ROS Robot Operating System.

ROV Remotely Operated Vehicle.

SGD Stochastic Gradient Descent.

SLR Sign Language Recognition.

TSL Turkish Sign Language.

ViT Vision Transformer.

List of Symbols

Chapter 1

Introduction

1.1 Motivation

Over the recent years, significant technological advancements have revolutionized the marine sectors. Extensive research in these fields has led to the improvement of underwater sensing and imaging instruments [1], the implementation of observation arrays and platforms [2], the increase of autonomy in platforms [3], and the development of new uses for marine robots [4] [5]. These advancements have also allowed underwater vessels to be successfully deployed to perform autonomous tasks such as seabed mapping [6], seafloor seismic observations [7], hazardous oil spills detection [8], and ships structure inspection [9]. Nevertheless, despite these achievements, certain underwater tasks, particularly those requiring human expertise like underwater maintenance, cannot be effectively performed by current levels of robotic autonomy.

Divers performing these kinds of activities usually operate in harsh and difficult to monitor environments with many risks where there is little room for malfunctions, lack of adaptability, or general lack of attention. A mishap regarding any of these issues may result in disastrous consequences. Divers are also required to move around in complex 3D environments, where they may need to carry specialized equipment to perform tasks such as inspection, maintenance, or underwater welding. Given the risks and dangers of marine operation, it is vital to develop tools and systems capable of aiding and supporting those who perform daily operations in this field.

Even though they may not accomplish the same tasks at full autonomy, Remotely Operated Vehicles

(ROV) and Autonomous Underwater Vehicles (AUV) can help to assist and reduce dangers for divers in many scenarios. Environments that require complex manipulation and situational awareness have a high potential for human-robot collaboration. Some examples in which this applies are archaeology, marine science, oil and gas production, and inspection and maintenance of marine infrastructure.

The research into the feasibility of underwater human-robot interaction to enable the deployment of marine robots to assist divers has intensified over the last decade [10]. The two most prominent underwater gesture recognition systems can be found as part of the CADDY project [11] and the RoboChatGest [12] implementation, both focusing on the classification of static gestures. The goal of this research is to build upon these works and develop an artificial intelligence based framework capable of detecting, classifying, and sending a robot a command in response to gestures performed by a human diver below the surface.

1.2 Objectives

The main objective of this work is to develop a system capable of *understanding* dynamic gestures, within a previously defined limited set, and send basic motion commands to a ROV/AUV. This will be done by exploiting recent advancements in computer vision and neural network models. In order to achieve this goal, the following specific objectives will be followed:

- Review the most renowned gesture recognition methods, both underwater and outside, to evaluate their weaknesses and strengths.
- Propose and build a baseline model and architecture to achieve our main objective. Initially, test the model on publicly available datasets.
- Design and record a dataset of underwater dynamic gestures, since there are no currently available datasets of dynamic diver gestures below the water. Those available only contain static gestures.
- Build a pipeline to perform motion control tests of an ROV vehicle upon reaction to detected gestures. The pipeline can be used for simulated and actual vehicles.
- Review the performance of the integrated system and propose improvements to the dynamic gesture recognition framework.

1.3 Contributions

The main contributions of this work can be summarized in the following items:

1. Implementation, improvement and training of a dynamic gesture recognition framework.
    (a) Changed pose detection paradigm of a baseline detection pipeline for increased accuracy and speed.
    (b) Implemented a new method of hand extraction, which is independent of an estimated depth, unlike the baseline method. This implementation has also been stabilized by implementing a window-smoothing approach to the key-point extraction.
    (c) Implemented video augmentation during training to improve robustness and generalization of the model.
2. Creation of an in-house underwater dynamic gesture dataset.
    (a) This dataset contains 20 gestures commonly performed by divers.

   (b) It is made up of approximately 500 video samples.
   (c) It contains a subset of divers with gloves on, a low-contrast situation that pushes the limits of hand detection.

3. Integration of the gesture recognition module with basic motion primitives for a BlueROV2 Heavy vehicle.

4. Implementation of a Finite State Machine to perform live dynamic gesture detection with a simulated environment. This environment is robust enough that it contains all components of an actual system. As such, it would be possible to use the developed code on an actual vehicle without any changes.

1.4 Outline

This work is divided into the following chapters:

- Chapter 2 introduces the context, background and state-of-the-art references used in following chapters.
- Chapter 3 provides a detailed overview of the dynamic gesture recognition framework to implement.
- Chapter 4 contains the results and discussion of the framework implementation. This includes each individual module of the framework and the architecture. It also includes results from the integrated system.
- Chapter 5 summarizes the most important parts of the work. And puts forward proposals for improvements and future work.

Chapter 2

State-of-the-Art

2.1 Human-Robot Interaction

Over the last couple of decades, robots have steadily increased their presence in manufacturing, science, medical and even household applications. However, these traditional systems are usually designed as isolated machines to keep humans outside of the loop; both for safety and standardization. However, over the recent years there has been a new category of collaborative robots being implemented. They are hailed by their potential for flexibility and capacity to blend in with their environments.

Robots ought to adjust to the existing communication methods employed by humans, utilizing voice in ground/air settings. The equivalent in the underwater setting would be comprehending gestures underwater, as commonly practiced by divers. Many research projects have been developed regarding the identification and interpretation of human issued commands, including voice-issued command recognition using artificial intelligence [13] [14]; the implementation of digital twins for the surrounding environment to use in virtual reality [15]; and the use of visual recognition models to identify human gestures and features [16].

2.1.1 Human-Robot Interaction in an Underwater Environment

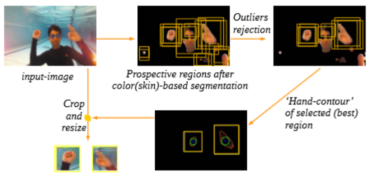
In an underwater setting, as mentioned by Birk, A. in [17], it is possible to split the Human-Robot Interaction problem in two parts: Robot-to-Human Communication (RHC), and Human-to-Robot Communication (HRC). While HRC has been the main research interest of these two, there have been remarkable studies and approaches to RHC.

Birk, A. discusses three main alternatives for underwater robot to human communication. The simplest feedback a human can receive from an ROV or an AUV is a short light signal, or a sequence of these. Most vessels are equipped with some sort of light source that can be used as a base for this communication. However, if a complex message needs to be transmitted, the human receiving the message needs to be thoroughly trained to decode it, opening the possibility to miscommunication in a setting where feedback would not be an option. Another option for RHC is the use of screen displays. This approach allows for higher expression capabilities and is closer to what we use every day, familiarity can be easy to exploit to generalize this approach. Lastly, Birk, A. mentions the possibility of using robot motion to transmit basic messages to the human. A small oscillating pitch rotation can be used as a 'yes' message, mimicking the nod of a human. Similarly, a small oscillating yaw rotation can be used to mimick a 'no'. This approach also exploits the familiarity of everyday gestures.

As for HRC, there have been a wide range of possible solutions researched into. The European Union CADDY (Cognitive Autonomous Diving Buddy) [18] project team, for example, developed a watertight tablet capable of being mounted on an underwater vessel and receiving diver inputs through the use of an inductive pen. This method facilitates the use of familiar techniques to humans, but requires very close proximity between the diver and the robot. One of the most common communication solutions is the use of human gestures to communicate with the robot. Gestures are already heavily used by divers in everyday tasks, so building upon this becomes a natural extension to their already existing language. Based on this, the CADDY team developed a sign language for HRC: Caddian [19]. This language contains both static and dynamic gestures, and its use is based on the compounding of these gestures to construct a message. This message or sentence needs a *start* gesture prepended and an *end* gesture appended to signal its beginning and end.

The underwater gesture recognition problem has been explored by several different approaches and combined with technology for different kinds of data capture, some of which can be observed in Figure 2.1. A classic approach is the use of cameras and computer vision techniques to capture and process dynamic gestures of the arms and hands. Complementing these techniques, some teams have relied on the use of specialized gloves to facilitate the visual detection of gestures [11]. This is specially helpful because divers may use gloves to keep their hands warm on certain situations. These gloves are usually black and they do not contrast well with diver suits, which are also mostly black. The identification of gestures is difficult with this color combination and the reduced visibility underwater. Nonetheless, there have also been implementations without the use of any particular wearable device. Islam, M. [20] has implemented a neural network to identify hand gestures that exploits the characteristics of human physiology for an improved detection; specifically, the algorithm detects the color of the skin to extract hand features. As far as visual approaches go, performing hand detection with gloves is an unusual task that provides many challenges. A different approach is the use of multi-beam sonar imaging, instead of a camera, to minimize the issues with underwater optics and grasp on the characteristics of acoustics [21]. Every proposal has benefits and disadvantages and there is no clear stand out solution to date.

(a) Specialized gloves developed for the CADDY project to facilitate visual detection.


Outliers rejection
input-image
Prospective regions after color(skin)-based segmentation
Crop and resize
'Hand-contour' of selected (best) region
(b) Skin segmentation and hand contour procedure performed by the RoboChatGest model to detect hand positions.

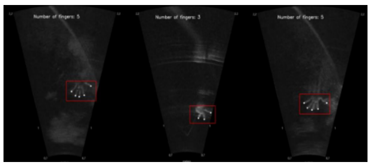

(c) Hand and finger detection using acoustic image from a multi-beam sonar.

Figure 2.1: Highlights from the mentioned methods for underwater HRI.

2.2 Human Pose Estimation

Many machine learning and computer vision tasks require 2D human poses as their input. It is a key component to enable a better understanding of humans in the context of images and videos. However, this information is not easy to acquire. Pose information of humans is not readily available in images captured with a camera, but this information is embedded in the image. As such, several approaches and neural network models have been developed to tackle the problem of human pose estimation from images.

Human pose estimation aims to locate anatomical keypoints of a human, either in a 2D or a 3D setting. This computer vision task becomes challenging due to the possible variations in the occlusion, truncation, and scales of the human in the image; aside from the particular differences between human appearances. This task becomes increasingly challenging when the gear and equipment a diver has to wear adds noise that is not present in most scenarios above water.

As stated by the survey on human pose estimation (HPE) models performed by Zheng, C. et al. [22], these models can be classified according to the technique they implement on their pipeline: regression methods and heatmap-based methods. Recent works have shown that, regarding regression models, features that encode rich pose information are critical to achieve good results. It has also been shown that multi-task learning generally allows models to learn better feature representation when trained for related tasks. Heatmap-based models, when compared to regression models, usually provide richer supervision information, since they preserve spatial information that facilitates the training of convolutional neural networks.

The number and locations of the keypoints to be estimated is defined by the human body model used for reference and annotation. The MS-COCO (Microsoft Common Objects in Context) [23] dataset annotates a maximum of 17 keypoints in the human body, while the MPII [24] dataset uses 16 keypoints. The AI Challenger Human Keypoint Detection [25] and CrowdPose [26] datasets both use 14 joint keypoints in their annotations.

In addition to these *basic* human body models, there are also specialized pose models for some specific body parts. For example, the OneHand10K [27] and InterHand2.6M [28], datasets specialize in hand pose keypoints. Both of these datasets infer 21 keypoints for each detected hand. However, OneHand10K works with 2D coordinates, while InterHand2.6M works with 3D coordinates. Similarly, whole body annotations are also available. These merge the body part specific joint keypoints to the basic human body models. The COCO-Wholebody [29] benchmark serves as an example of such datasets.

Classic top-down [30] approaches to human pose estimation, like Mask R-CNN [31] and Cascaded Pyramid Network [32] models, tackle this problem by starting with a person detector. Given a single image, the task of this detector is to identify the number of people in the image and their location, as bounding boxes. Each identified person and bounding box is then fed into the pose estimator for inference. This approach has proven to be useful and successful in many cases, despite its simplicity. However, since it performs operations on a per-human basis, the computation and processing time is directly proportional to the number of identified people in the image, resulting in slow performance when there is a large number of individuals in the image. This approach also propagates forward any noise that originates during the early detection phase; a misdetection (or lack of detection) will input the wrong information to the pose estimator, which will infer an incorrect pose.

On the other hand, bottom-up approaches, e.g. OpenPose [33] and XNect [34], bypass the need for an early detection phase and are able to perform inference on the raw input image. While this removes the possibility of early misdetections, it also means the estimator model will receive any noise the detector may have filtered otherwise. These approaches do not suffer from increased processing time as the number of people increase in the image, but will have a processing time linearly proportional to the resolution of the raw image.

2.2.1 OpenPose

OpenPose was developed in 2016 looking for a solution to the issues observed in top-down approaches. The main body detection feature of OpenPose uses a bottom-up approach based on association scores via Part Affinity Fields. These are a set of 2D vectors that encode the location and orientation of body parts on the image. Using this method, the estimator can identify the keypoints for ankles, knees, hips, shoulders, elbows, wrists, necks, torsos, and head tops. Part Affinity Fields also help to eliminate false associations between limbs not connected to each other. A comparison of inference time between OpenPose and contemporary top-down models can be observed in Figure 2.2.

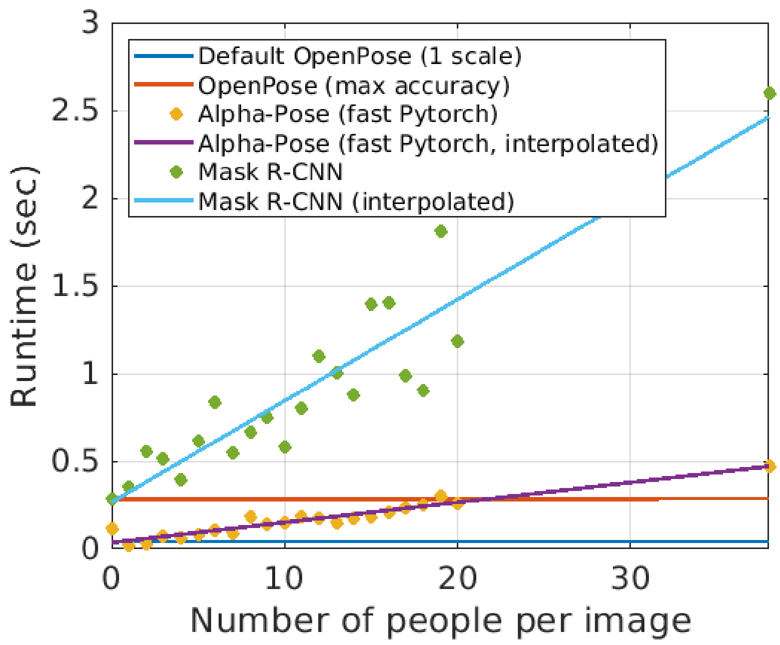


Figure 2.2: Inference time comparison between OpenPose (bottom-up), Mask R-CNN (top-down), and Alpha-Pose (top-down). The inference time of OpenPose's bottom-up approach is invariant to the number of people in the image, while the top-down approaches grow linearly with the number of people. Figure extracted from [33].

2.2.2 MediaPipe

Google's MediaPipe also offers a suite of libraries, tools, and solutions for machine learning development. Amongst their solutions, they include vision tasks like object detection, image classification, image segmentation and pose landmark detection. This is a ready-to-use optimized toolbox that offers good results, but may not be as flexible as other relevant toolboxes in the field.

Their pose landmark detection solution identifies key points on a human body in an image or a video. The output of this task is given in both image coordinates and 3-dimensional world coordinates. Their pose landmarker model identifies 33 body landmark locations. This includes key landmarks in the face, hands, feet, and body.

2.2.3 MMPose

MMPose [35] is an open-source toolbox for pose estimation based on PyTorch. It contains several algorithms for 2d multi-person human pose estimation, hand pose estimation, face landmark detection, and animal pose estimation. MMPose supports both bottom-up and top-down approaches. For top-down approaches, it works in tandem with another library from the same lab, MMDetection [36], to identify person bounding boxes. More importantly, due to the facilities it offers regarding dataset and model management, it is used as a development base by several state-of-the-art pose-related algorithms, techniques, datasets, and model backbones.

Regarding data management, the MMPose toolbox facilitates the training and testing of pose estimation models on annotated datasets. To aid on these tasks, MMPose implements data processing pipelines that easily performs augmentations and transformations to input images. An example of a typical pipeline can be observed in the following code snippet:

```python
train_pipeline = [
    dict(type='LoadImage'),
    dict(type='GetBBoxCenterScale'),
    dict(type='RandomFlip', direction='horizontal'),
    dict(type='RandomHalfBody'),
    dict(type='RandomBBoxTransform'),
    dict(type='TopdownAffine', input_size=codec['input_size']),
    dict(type='GenerateTarget', encoder=codec),
    dict(type='PackPoseInputs')
]
test_pipeline = [
    dict(type='LoadImage'),
    dict(type='GetBBoxCenterScale'),
    dict(type='TopdownAffine', input_size=codec['input_size']),
    dict(type='PackPoseInputs')
]
```

Listing 2.1: MMPose data augmentation and transformation pipeline.

These pipelines turn loading and pre-processing different-sized images into a trivial issue. The MMPose toolbox predefines a set of transformations and augmentations commonly used in pose estimation training and provides them as easy-to-call dictionaries. This helps to avoid the task of defining transformations for a dataset, where you have to make sure all the dimensions match. This can be specially troublesome if you're working with different sized inputs and multiple models. Under the hood, MMPose assures there are no mismatches when you invoke the dictionary in the pipeline and use it as part of the dataloader. Moreover, the framework also facilitates the implementation of encoding and packing data.

Regarding model management, MMPose breaks down each model in four components: a data preprocessor, a backbone, a neck, and a head. The data preprocessor is in charge of performing data normalization and channel transposition, to better exploit massively parallel computational capabilities of modern Graphics Processing Units (GPU). The backbone is a known network that has been previously trained and used in similar tasks. The neck is an optional module between the backbone and the head; some commonly used necks are Global Average Pooling (GAP) and Feature Map Processor (FMP). Finally, the head contains part of the algorithm specific to the task. It dictates the the prediction and performs loss calculation. The modular approach to model management in this toolbox, makes it straightforward to switch out one of these components and compare the performance after this change.

This toolbox actively supports two of the most recent pose estimation models with great results on benchmark datasets: ViTPose [37] and RTMPose [38].

2.2.4 ViTPose

One of the most recent breakthroughs in neural networks and deep learning is the development of transformers, and more specifically, vision transformers. Yufei Xu et al. propose a simple baseline model to demonstrate the potential of plain vision transformers in human pose estimation: ViTPose.

ViTPose uses a top-down approach for human pose estimation built upon the MMPose toolbox. It is based on a detector to highlight human instances in an image on which ViTPose is used to estimate their keypoints. The implementation of the ViTPose model uses vision transformers as the backbone, pretrained and initialized on ImageNet-1K with the use of masked image modeling. Following this, the extracted features are processed by a decoder to upsample the feature maps and regress their heatmaps. The general overview of the ViTPose framework can be observed in Figure 2.3.

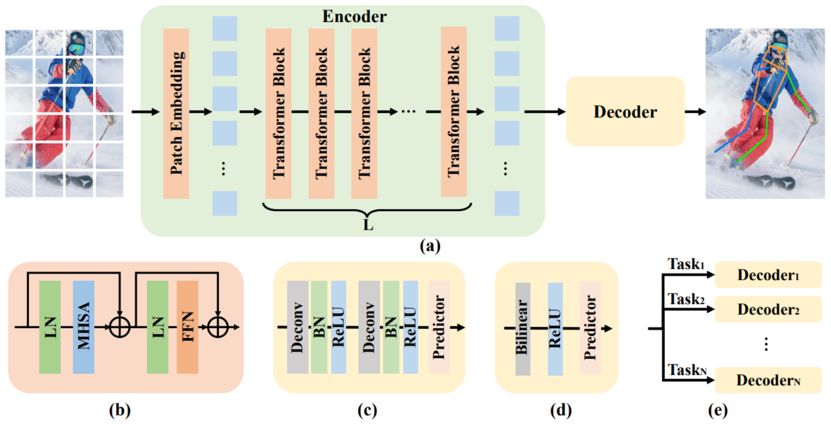


Figure 2.3: Highlights of ViTPose framework. (a) the framework of ViTPose, (b) the transformer block, (c) the classic decoder, (d) the simple decoder, and (e) the decoders for multiple datasets. Figure extracted from [37].

2.2.5 RTMPose

Jiao, T. et al. have developed a a real-time multi-person pose estimation framework while empirically improving key factors to enhance performance and reduce latency. Their work focuses on five aspects of pose estimation frameworks: approach or paradigm, backbone network, localization method, training strategy, and deployment. Their improvements on these aspects allowed their model to reach state-of-the-art accuracy and speed, as shown in Figure 2.4.

RTMPose follows a top-down paradigm, using an off-the-shelf detector to obtain human bounding boxes. The authors explain about top-down algorithms that, despite being classified stereotypically as

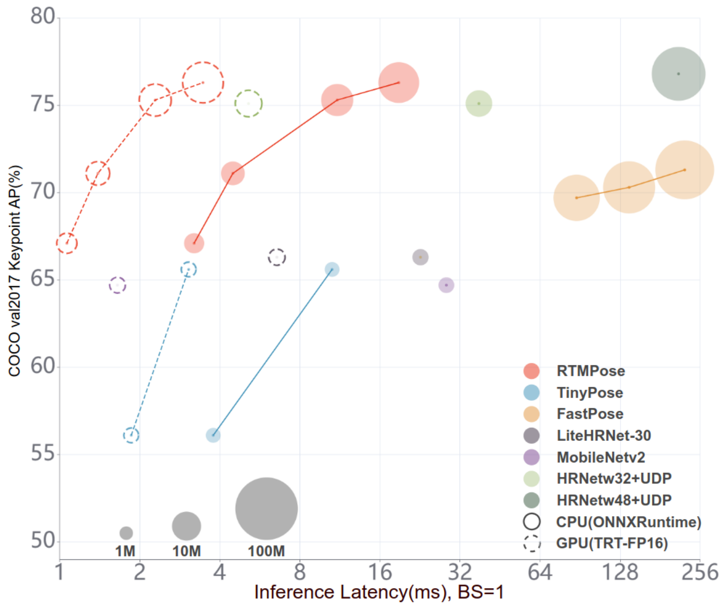
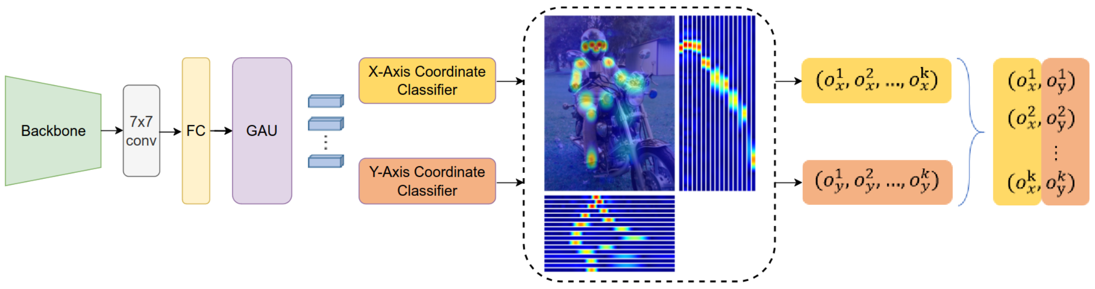


Figure 2.4: The comparison of RTMPose against open-source libraries on the COCO val2017 set; circle size represents the relative size of model parameters. Figure extracted from [38].



Backbone
7x7 conv
FC
GAU
X-Axis Coordinate Classifier
Y-Axis Coordinate Classifier
$(o_x^1, o_x^2, ..., o_x^k)$
$(o_y^1, o_y^2, ..., o_y^k)$
(o_x^1, o_y^1)
(o_x^2, o_y^2)
(o_x^k, o_y^k)

Figure 2.5: RTMPose's architecture. A convolutional layer, a fully-connected layer and a Gated Attention Unit (GAU) are used to refine K keypoint representations. After that, two classification tasks are done to perform 2d pose estimation. Figure extracted from [38].

slow, recent breakthroughs have achieved excellent speed and performance. On scenarios with less than 6 people in the image, Jiao, T. et al. state that their lightweight model is able to perform multiple forwards pass for every instance in real time.

Their proposed model uses CSPNeXt [39] as backbone. A key difference from usual pose estimation backbones is that CSPNeXt was designed for object detection, not image classification. Their chosen backbone offers a good balance between speed and accuracy by exploiting its proven ability to perform well on dense prediction tasks.

While most keypoint prediction frameworks use a heat-map algorithm to perform keypoint location predictions, the authors based their approach on the method showed in SimCC [40]. In simple terms, this approach breaks down the problem into two classification tasks: one for each 2D coordinate. The high-level operation of their proposed framework can be observed in Figure 2.5.

The authors also performed several training improvements on their approach. First, they implement a heatmap-based pre-training on the backbone. They also perform a series of changes to their optimization strategy, e.g. implementing exponential moving average and flat cosine annealing. They also perform two phases of data augmentations during their training to reduce the side effects of “noisy” samples in their model. These are but hand-picked examples of their long list of implemented improvements. Figure 2.6 contains a bigger list of the improvements performed.

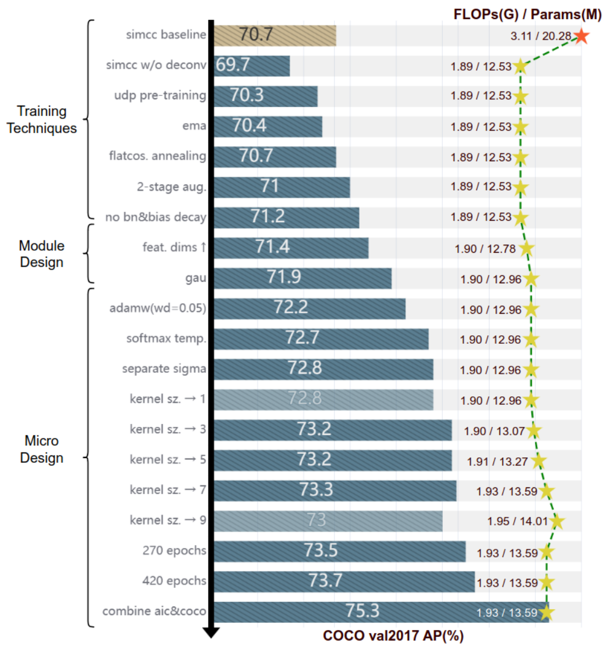


Figure 2.6: List of the improvements performed step by step by the authors and the results achieved at each step. Figure extracted from [38].

Their implemented modifications allowed them to achieve state-of-the-art performance on the COCO, COCO SinglePerson, COCO WholeBody, CrowdPose, MPII, and AP-10K [41] datasets.

2.3 Hand Pose Estimation.

Similar to human body pose estimation, human hand pose estimation is the task of finding the joints of the hand from an image or set of video frames. This allows the direct use of the hand as an input to many applications. Machine control, augmented reality and virtual reality are among the possible uses for these techniques. There have been numerous proposals for solutions, which can be typically divided into 2D keypoint localization and 3D keypoint localization. Some of the usual obstacles found in this problem are similar to those appearing in human pose estimation, such as occlusion, natural differences in hand shape and size, changing light conditions, and varying backgrounds.

Among the 2D solutions to this problem, Santavas et al. [42] propose a CNN architecture for direct hand pose estimation. Their implementation infers the coordinates of a hand's keypoint from a single RGB image. On their research, they make use of a visual attention mechanism to help the neural network to identify and focus on the more relevant attributes of an input. This allowed them to present a model that achieves competitive results while keeping a low complexity and a computationally efficient architecture. As mentioned on the previous section, hand pose estimation can also be included in whole body pose estimation.

In 3D hand pose estimation, most of the research is focused on using depth maps as input, since current 3D estimators based on images have proven to perform subpar. Among the most recent depth-based research, there are three methods that are achieving consistent good results in the main datasets related to hand pose estimation (NYU Hands [43], ICVL Hand Posture Dataset [44], HANDS 2017 [45], and HANDS 2019 [46]). Cheng J. et al. [47] propose a method to augment the depth map and create multiple viewpoints for the same input. Their approach fuses the estimated hand poses from each of the virtual viewpoints, the result manages to accurately and robustly resemble the ground truth 3D hand pose. Figure 2.7 shows a basic example of the generation and selection of virtual viewpoints. Rezai M.

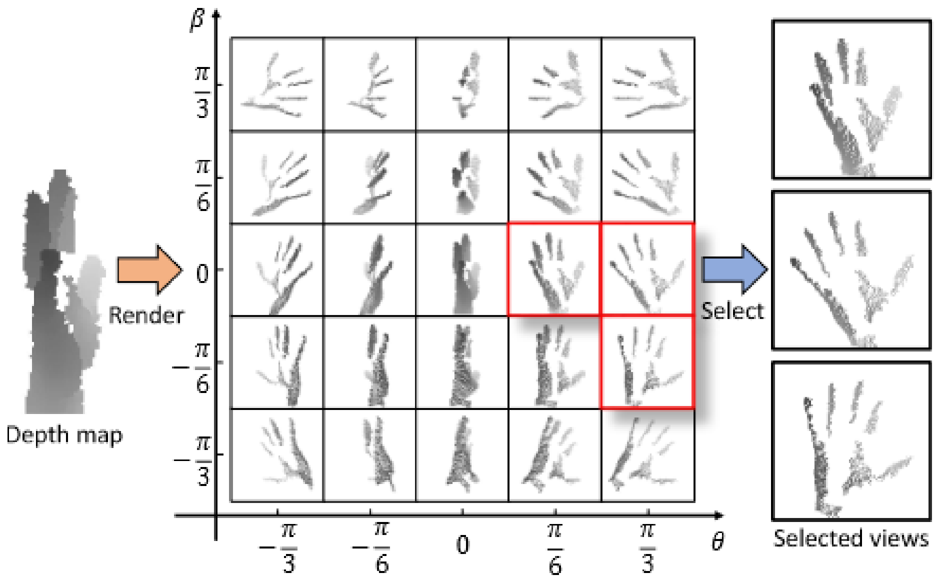


Figure 2.7: Illustration of multiple virtual views generated and selected for 3D hand pose estimation. Figure extracted from [47].

et al. [48] propose TriHorn-Net, a new neural network model for accurate depth-based 3D hand pose estimation. TriHorn-Net achieves better accuracy by decomposing the 3D hand pose into a 2D estimate of the joint location and an estimate of its depth value. Their neural network architecture first encodes the depth map into high-resolution features. This is then fed into three parallel branches: the 2D joint coordinate estimator, the attention enhancement module, and the depth estimator branch.

2.3.1 Hand pose estimation in toolboxes

Aside from the aforementioned novel approaches on 2D and 3D hand pose estimation, several well-established pose estimation toolboxes include some capability to perform hand pose estimation.

The OpenPose library includes such an implementation of hand pose estimation. However, unlike the estimation they perform on their body model, the hand pose estimation is performed following a top-down approach by using a detector to locate the hands in an image. Once all the hands are located, pose inference is performed on each of the identified instances. As a consequence, the runtime of this model is no longer invariant to the number of detected people.

The MediaPipe toolbox also includes a solution for hand pose estimation. Their hand landmarker model identifies 21 key points of hand-knuckle coordinates. This model is trained on around 30,000 real images and several synthetic images with background variations. This solution is bundled with their palm detection model, which locates the position of the hands in the images before the keypoint estimation is performed. Their approach is optimized to take advantage of the information obtained in previous frames and use it on the current frame. This model also delivers a handedness estimation for each processed hand; it tries to detect if it is a left hand or a right hand.

The MMPose open-source toolbox also contains capabilities of using hand model configurations and running hand pose estimation in a top-down approach. Most of the models implemented on top of this library are able to be trained on different body-part models to observe their performance under different scenarios. Because of this, newly developed models like ViTPose and RTMPose have available ready-to-use checkpoints for hand pose estimation.

2.4 Static Hand Gesture Recognition

The hand gesture recognition task refers to the classification of movements and shapes performed by human hands. Humans are capable of easily recognizing these due to their combination of vision and synaptic interactions developed in the brain. Solving this task for a computer requires the ability to differentiate objects of interest in the image, process these objects properly and perform some classification afterwards. Static gestures refer to those cases where a single image needs to be processed to properly classify the gesture. In the following subsection, a static gesture recognition approach for underwater images will be analyzed.

2.4.1 RoboChatGest

In a direct underwater setting, Islam, M. et al. [12] developed a real-time task programming and parameter reconfiguration method for AUVs using static hand gestures. They performed the recognition with a CNN focused on the area around the hands. Hand detection is performed via 4 steps: segmentation, hand contour matching, outlier rejection and selection. Initially, the whole image is blurred and a threshold is set in HSV for skin-color segmentation. This step helps narrow down the possible areas in the image where a hand may be located. After all possible areas are identified, each different region is contoured and its intrinsic parameters are determined. Outlier rejection is performed by exploiting cached information regarding the hand gestures detected in the previous frame. Then, the possible hand contours are matched with a bank of previously defined class contours. The selected region is the one most closely related to one of the classes. Left and right hand regions are both independently selected in parallel during this step. Finally, the region is cropped and resized to be processed in a Convolutional Neural Network.

The classified gestures are then processed in a finite-state machine. This is used for efficient gesture-to-instruction mapping. An additional constraint requiring the gesture to be detected for 15 consecutive frames is included. The idea behind this implementation is to reduce spurious gesture recognition and improve robustness of the method. The complete framework can be observed in Figure 2.8.

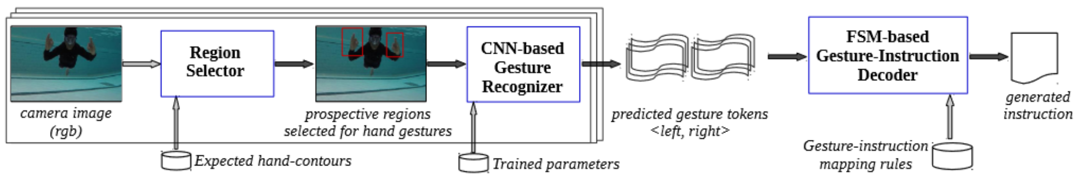

camera image (rgb)
Region Selector
prospective regions selected for hand gestures
CNN-based Gesture Recognizer
predicted gesture tokens <left, right>
FSM-based Gesture-Instruction Decoder
generated instruction
Expected hand-contours
Trained parameters
Gesture-instruction mapping rules

Figure 2.8: Highlights of RoboChatGest framework. The left half shows the region selector and the CNN. The right half depicts the finite-state machine for robust mapping gestures to instructions. Figure extracted from [12].

Using this method the authors were capable of relaying a simple set of commands to an AUV. However, there are a couple of significant limitations present on this method. First, the available gestures for classification are very limited. Based on the dataset used for training and the communication approach, the CNN only identifies 10 different static gestures. Secondly, there is margin for improving the detection and identification of hand gestures; frames may be skipped over and similar gestures may be mixed.

2.5 Dynamic Hand Gesture Recognition

Dynamic hand gesture recognition brings in additional challenges in comparison to its static counterpart. This problem is defined by the need of processing a sequence of images to properly classify the gesture.

It requires a more complex approach, taking into account the variations in time and the use of different tools and models. In the following subsections, two dynamic gesture recognition approaches will be explored.

2.5.1 StaDNet

O. Mazhar et al. [49], propose an unified framework that recognizes both static and dynamic gestures in images acquired from a monocular camera. The result of their work is StaDNet, a neural network architecture coupled with a spatial attention module to pre-process the inputs sent to the classifier model. StaDNet is divided in three parts: a pre-processing Spatial Attention Module, the Static Gesture Classifier and the Dynamic Gesture Classifier. A quick overview of the method will be done in this section, but a more detailed description can be found in Chapter 3.

The Spatial Attention Module contains a human pose estimator and three fully-connected neural networks to estimate the depth of the body, the right hand and the left hand. It also performs data augmentation on the estimated pose, which is proven to improve the prediction results of neural networks. Finally, the hands are also located and an image for each hand is cropped in this module. These images, focused around each hand, will be part of the information fed into the main neural network for gesture recognition.

The Static Gesture Classifier takes both cropped hand images and processes them with a CNN. The authors utilized a pre-trained Inception V3 model as backbone, as this was state of the art on image classification when they published their research. They fine-tuned the model to classify 9 different gestures from American Sign Language (ASL) and an additional *None* gesture. Each image frame and hand is processed independently.

The Dynamic Gesture Classifier receives a fused vector as an input. This vector contains embedded information extracted from the two hands via the Static Gesture Classifier and also contains the augmented pose vector computed on the Spatial Attention Module. This vector is processed through several fully connected layers and Long Short-Term Memory (LSTM) [50] blocks to classify the dynamic gesture being represented.

The method implemented in [49] uses RGB images as input, due to the use of OpenPose. If the pose estimation method were to be compatible with HSV, the input could be modified to this color model for increased robustness. Moreover, the authors also mention that while their work was mostly performed under ideal lighting conditions, data augmentation strategies could prove useful to improve the performance under extreme lighting conditions.

2.5.2 Dynamic Graph-Based Spatial-Temporal Attention

Chen, Yuxiao et al. [51], propose a Dynamic Graph-Based Spatial-Temporal Attention (DG-STA) method to perform hand gesture recognition. They state that even though recent research has achieved significant improvement with deep learning, usual methods exclusively feed the neural network a tensor containing the information of the joint coordinates. However, this input does not exploit the spatial structures and temporal dynamics of hand skeletons. Prior to their work, and to correct this shortcomings, some authors have implemented pre-defined spatial-temporal graphs to embed the structures and dynamics of hands, but these kind of approaches lack flexibility when capturing different set of actions. The DG-STA implementation in [51] attempts to address this lack of flexibility by creating an adaptive graph to successfully capture the specific features of different actions.

The implementation of the DG-STA model performs hand gesture recognition in 4 steps. First, they construct a fully connected hand skeleton graph from the input of the joint coordinates for the hand. Second, they extract the spatial-temporal information from the hand skeleton graph. Third, they combine

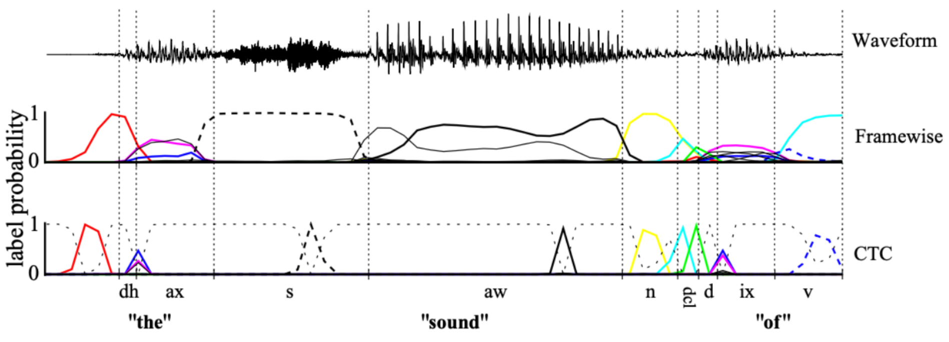


Figure 2.9: Example of a CTC network classifying a speech signal. Figure extracted from [52].

the spatial-temporal features with the features extracted from each node to create an embedding with enriched information. Fourth, they implement a spatial-temporal mask to increase the efficiency of the model.

2.6 Gesture Identification

A related problem with dynamic gesture recognition is the identification of a gesture's length and location within a sequence. That is, given a long sequence of information, determine where a gesture is beginning and where it ends. This problem usually does not exist when developing dynamic gesture recognition models, as the information is pre-processed to contain only the sections of interest within a sequence. However, it must be solved when such models are to be deployed for general use, where the input information will not be preprocessed for the model; otherwise the applicability of gesture recognition model would be limited. This is an issue that has already been explored in other areas of application, speech recognition for example. One of the successful solutions for this issue on speech recognition will be lightly touched on this subsection, the Connectionist Temporal Classification (CTC) [52], and a more rudimentary approach will be established for visual gesture sequences.

2.6.1 Connectionist Temporal Classification

The development of this approach was born of the need to perform predictions of sequences of labels from noisy, unsegmented input data. It calculates a loss between a continuous unsegmented time series and a target sequence. This is performed by summing over the probability of possible alignments, resulting in a differentiable loss value for each input node. By implementing this loss, a network can be trained to successfully detect the instances of interest in a sequence. In Figure 2.9, a CTC-trained network can be observed performing classification of sequences of phonemes for speech recognition.

2.6.2 Static Gesture Mask

A much more naive approach, that does not seek to implement an automatic detection of gestures of interest, is the use of a simpler model before performing dynamic gesture recognition. The goal of this static gesture mask is to detect two gestures: one that marks the beginning of a dynamic gesture and another one that marks the end of the dynamic gesture. A similar approach was successfully used during the Caddian language tests.

2.7 Artificial Neural Networks

All of the mentioned methods for body pose estimation, hand pose estimation, static gesture recognition, and dynamic gesture recognition use some type of artificial neural network (ANN) in their framework for one or more tasks. Artificial neural networks are a subset of machine learning and the core of deep learning algorithms. They are inspired by the human brain and seek to mimic the method by which human neurons communicate with one another.

ANNs are typically made up of an input layer, one or several hidden layers, and an output layer. Each one of these layers consists of a number of nodes, which are also called neurons. In the most basic situation, each node is connected to every node of the following layer and possesses a weight, a bias, and a threshold needed for activation. If the output of an individual node is above the threshold, it is activated and sends information forward. The general layout of a basic fully-connected ANN can be observed in Figure 2.10. These neural networks are usually trained with annotated data related to a specific task to achieve a good performance before deploying the model. During training, models learn the best weights and biases using optimization methods. The optimization is based on an error or loss function that describes the gap between the current results and the expected ones.

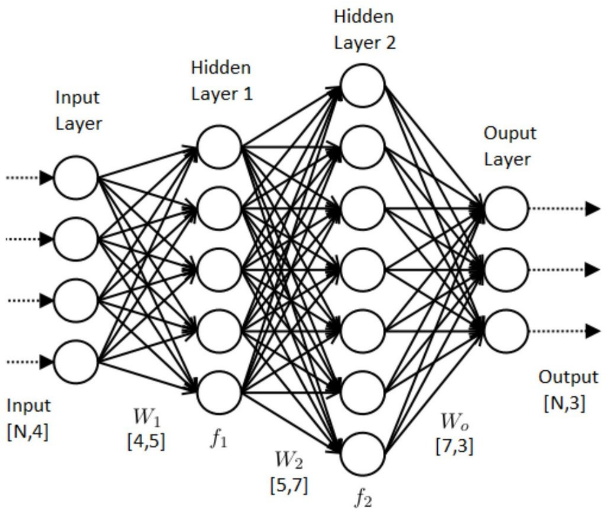


Figure 2.10: Example of a 2-hidden-layers artificial neural network.

Several improvements and innovations have been developed in artificial neural networks since the introduction of the basic scheme observed in Figure 2.10. The sum of these improvements now allow us to implement neural network models capable of image classification, speech recognition, object detection, pose estimation, among other tasks. Two of the most common types of neural networks used nowadays are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). The basic idea of these will be explained in the following subsections.

2.7.1 Convolutional Neural Networks

Convolutional neural networks specialize in processing data structured in a grid-like fashion, like images. They are characterized by the introduction of the convolutional layer, which have become the building blocks of every CNN model. In simple terms, they perform a dot product between matrices A and B. Where matrix A is the kernel of the layer, a set of learnable parameters. And matrix B is the receptive field, which is given by the size of the filter in the convolutional layer. This filter slides sequentially along the input grid, thereby allowing to make use of spatial relations in the data. Convolutional blocks have

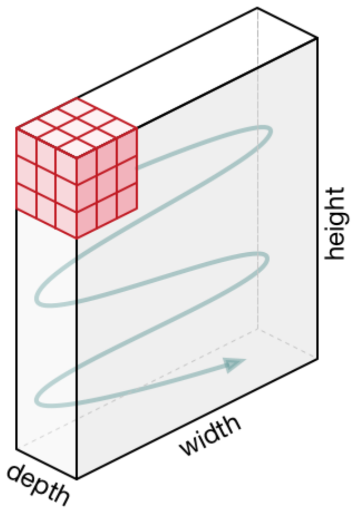


Figure 2.11: Example of kernel movement in a CNN layer.

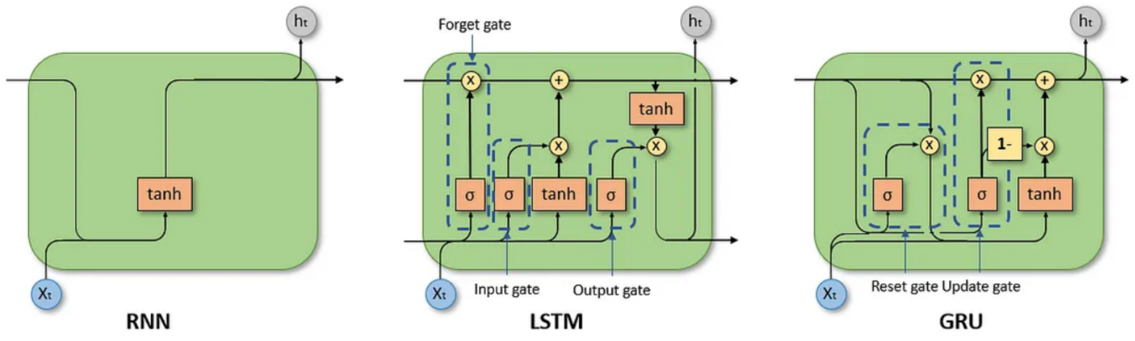


Figure 2.12: Diagram of a basic RNN, an LSTM cell, and a GRU cell.

proven to be very effective at capturing and learning features of varying complexity at different depths of a network.

2.7.2 Recurrent Neural Networks

Recurrent neural networks specialize in learning from sequential data. They are commonly used for sequential problems, such as natural language processing and speech recognition. They are known for exploiting their memory capabilities, which allows these units to take information from prior inputs and merge it with current inputs and influence their outputs. These networks can be unidirectional or bidirectional. Unidirectional RNNs work in a feedforward fashion, meaning the information flows following the sequence of the data. In bidirectional RNNs, the information is also enabled to work backwards, which allows the use of future data on the processing of the current time step.

The modern foundation of current RNNs models are based on the inception of key cell architectures. The most popular RNN architecture blocks are the LSTM and the Gated Recurrent Unit (GRU) [53].

Cahuantzi, R. et al. [54] perform a brief, but thorough, comparison between LSTM and GRU networks for learning symbolic references. During their study, they obtained results that show that an increase in RNN depth increases the training but does not necessarily increase the forecast accuracy. They also state that GRU networks generally outperformed LSTMs in low-complexity strings. However, the opposite was found on high-complexity strings, where LSTM networks had better results. Regarding hyper parameters, they found that the learning rate and the number of units per layer had the most significant influence on the results.

Motion Primitives		
No	Motion	Expected BlueROV Behaviour
1	Up	moves upward towards the surface.
2	Down	moves downward towards the seafloor.
3	Left	moves to the left.
4	Right	moves to the right.
5	Forward	moves forward, towards the diver.
6	Backward	moves backward, farther from the diver.
7	LookUp	tilts upward.
8	LookDown	tilts downward.
9	LookLeft	tilts to the left.
10	LookRight	tilts to the right.
11	Rotate	rotates 360° around itself.
11	RotateAroundToLeft	fixates on the diver and rotates around it, from the left.
12	RotateAroundToRight	fixates on the diver and rotates around it, from the right.
13	Follow	fixates the current distance to the diver and keeps it.
14	Hover	stays in place.
15	Stop	stops current motion.
16	Surface	goes to the surface.

Table 2.1: List exemplifying possible motion primitives for the BlueROV.

2.8 Underwater Vessel Motion Primitives.

A motion primitive is a simple predefined motion that a robot can perform in any given situation and that can be combined to execute more complex maneuvers. In the underwater setting, and referring an underwater vehicle, the available motions are determined by the actuators and the degrees of freedom available. The BlueROV2 Heavy, for example, possesses 8 actuators, providing a complete control over the six degrees of freedom while underwater. Some basic motion primitives can be seen in Table 2.1.

Note that this table does not list all the possible motions available to the BlueROV Heavy, it is only a small sample of the most common motions that can be implemented for testing the proposed human-robot cooperation methodology. Additional and more complex motion primitives can be added depending on the context.

If the underwater vessel were to have a payload or an actuated limb, similar motion primitives could be established for it. For example, ’Sample Now’ to perform measurements, ’Extend Arm’ or ’Close Clamp’ for the motion of an arm. In a similar fashion, *virtual primitives* can also be established. These primitives would not result in a direct physical action, but could work as a trigger for digital actions such as taking a snapshot or signaling an alarm to a colleague.

2.9 Diving Gestures

Regular communication becomes an issue underwater, where typically one cannot use common and hands-free methods such as speaking. Over the years, the scuba diving industry has developed over hundreds of hand gestures to facilitate underwater communication. These gestures help navigation and safety. Most of these gestures are universally used and lead the way for possible innovations in underwater HRI.

Some of these gestures can be static shapes with your arms or your hands, while others require limb movement. “Follow me”, “Stop”, “I’m cold”, and “Low on air” are a couple of examples meanings and commands the gestures in this language can transmit. A good understanding of these gestures is key to the success and safety of underwater activities. In Figure 2.13, some of these gestures are exemplified.

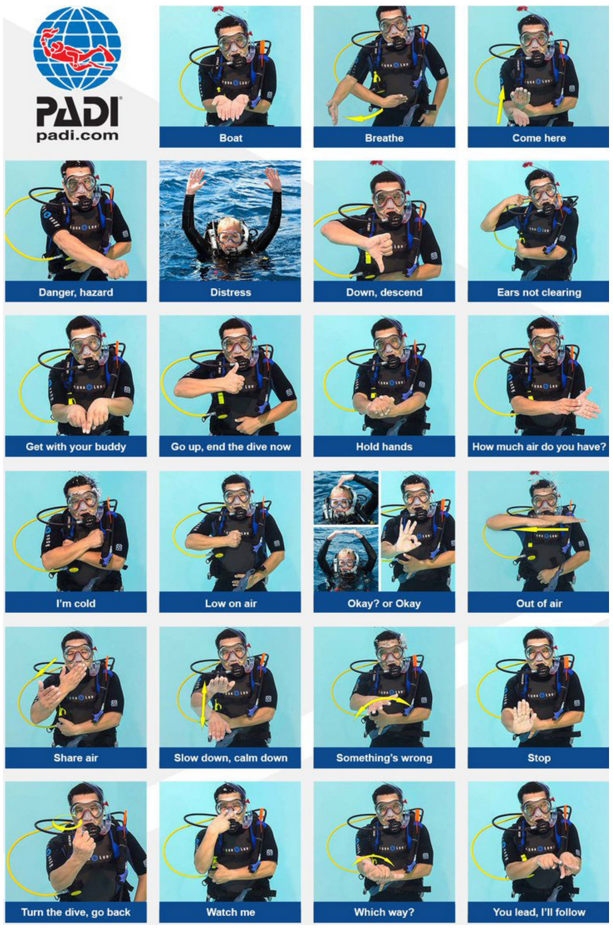


Figure 2.13: A set of underwater gestures scuba divers use underwater. This set contains both static and dynamic gestures.

Since current state of the art approaches do not tackle, or have not been able to perform, dynamic gesture recognition underwater a successful recognition through artificial intelligence would be a big step towards the exploitation of these communication channels to perform diver-AUV cooperative tasks.

Chapter 3

Methodology

In this chapter, the overall methodology to perform underwater hand gesture recognition is motivated and detailed. This section will cover the proposed neural network architecture for this task, the algorithm proposed to improve the stability of human pose estimation, the required pre-processing computations of the input image, the mapping of classified gestures to motion primitives, and general information about AUV motion control.

The implementation of this project is based on the StaDNet architecture, which is presented in section 2.5.1. It implements a stable pose estimator that can be exploited to perform robust hand identification and it also employs a data augmentation method on the pose estimation to improve the quality of the information in each frame. A detailed illustration of the StaDNet framework can be observed in Figure 3.1.

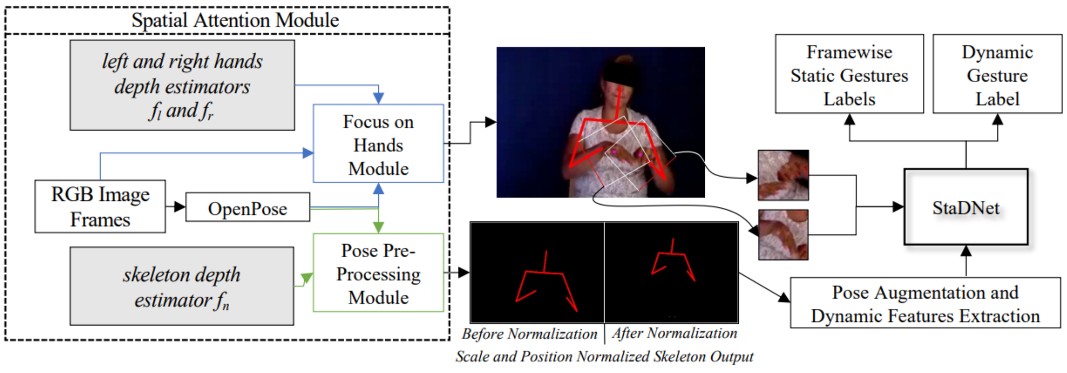

Spatial Attention Module
left and right hands depth estimators f_l and f_r
Focus on Hands Module
RGB Image Frames
OpenPose
Pose Pre-Processing Module
skeleton depth estimator f_n
Before Normalization
After Normalization
Scale and Position Normalized Skeleton Output
Framewise Static Gestures Labels
Dynamic Gesture Label
StaDNet
Pose Augmentation and Dynamic Features Extraction

Figure 3.1: Highlights of StaDNet framework. The Spatial Attention Module contains three learning-based depth estimators, the Focus On Hands Module (FOH) and the Pose Pre-Processing Module (PP). A 2D skeleton is extracted with OpenPose. FOH crops the hands by exploiting the keypoints from the skeleton and the depth from the hand depth estimators. Figure extracted from [49].

The first section of the StaDNet framework, the Spatial Attention Module, includes four neural network estimators (one pose estimator and three depth estimators), the focus on hands module, the pose pre-processing module, and it performs all the processing of the camera input before feeding it to the main neural network. Human pose estimation is performed on the raw image, which will be further explained in section 3.1. This step is key to track the movement of the skeleton along several frames, and to normalize the size and position of the subject. The estimated pose is also augmented by calculating additional vectors and angles from the base estimated joint keypoints. The method to augment the pose vector will be detailed in section 3.2. This augmentation setup is important for their implementation to provide an accurate estimation of the three depths: the depth of the neck, which serves as a reference of the depth of the body and is used for normalization; the depth of the right hand, which will be used to identify the scale and the size of the cropped region around the right hand; and the depth of the left hand, which works like the previously mentioned right hand. In our implementation, however, the only depth to be used is the one estimated at the neck joint. The methodology used for the depth estimator will be detailed in section 3.3.

As previously stated, the Focus on Hands Module uses the coordinates of the hands in the estimated pose to locate the center of each hand. Then, this location is processed along with the estimated depth of each hand to build a bounding box around each hand. This bounding box is used to crop the image and effectively extract the hand and the immediate surroundings. This is performed independently for each hand. In our implementation, the goal of the Focus on Hands Module remains exactly the same, but is achieved through a different method. This will be further explained in section 3.4.

The Pose Pre-Processing Module is divided in three steps: skeleton filter, skeleton position and scale normalization, and skeleton depth estimation. The goal of the skeleton filter is to rectify the occasional disappearance of joint keypoints that may hinder the training and the performance of the network. This filter works in a window of K consecutive images. In the tests performed by the authors, they achieved optimal results with a window of size 7; 3 images prior to the current frame and 3 images after it. This set of images and estimated poses are used to replace the coordinates of missing keypoints and Gaussian smoothing is applied to each joint keypoint over all images in the window. This reduces jittering in the pose and smooths joint movements. The skeleton position and scale normalization is performed to remove the influence of the user's position in the image and to remove the influence of the user's depth. Position normalization is performed by removing the keypoint coordinates of the neck from every other keypoint. Scale normalization is performed by dividing each shifted keypoint by the estimated depth of the neck. Once the skeleton is normalized in both position and scale, it is augmented as explained in section 3.2.

The second section of the StaDNet framework, the main neural network, receives three inputs: both cropped hand regions from the previous section and the augmented normalized pose keypoints. The architecture can perform both static and dynamic gesture recognition. Static recognition is performed by using the previously cropped hand regions, individually. Besides static gesture recognition, the processed hand images are also used for the dynamic counterpart. On dynamic gesture recognition, hand embeddings are extracted from the static gesture identifier and combined with the augmented pose keypoints. This vector is processed through a couple of fully connected layers and LSTM blocks before classification. A more in-depth explanation of this architecture is done in Section 3.6. The proposed model is trained on the ChaLearn Large Scale Signer Independent (LSSI) Isolated Sign Language Recognition (SLR) Dataset [55], from CVPR'21. This is a multimode dataset that contains isolated Turkish Sign Language (TSL) videos. Details of this dataset will be further explained in section 3.7.

Once the model is trained and ready to identify continuous gestures, the isolation of gestures from a longer video will be implemented. This will be done in a very simple fashion, by identifying opening and closing static gestures to mark the times of interest in the video. The explanation of this method will be detailed in section 3.9.

3.1 Robust Human Pose Estimation

This section of the dissertation further explains the Human Pose Estimation problem and the approach to solving it. Pose estimation is the task of, based on an input image, inferring the keypoint coordinates of the most important joints in the human body. Current state of the art pose estimators tend to present some variability when used for video outside of a controlled environment, as visual conditions may change very rapidly from one frame to the next. As such, the implementation of an algorithm to smoothen the response will be also proposed in this section.

3.1.1 Pose Estimation using Vision Transformers

A Vision Transformer (ViT) is a model for image-related tasks, such as image classification and segmentation, that focuses on attention mechanisms to draw global dependencies between inputs and outputs. It is an implementation of a transformer block for image processing. These kind of models were first introduced by Dosovitskiy et al. [56]. It proposes an alternate architecture to the more widespread CNN and RNNs. In ViTs, the input image is divided into several patches with a fixed width and height. These patches are converted into a linear embedding and a positional component is added to each, which refers to the position of each individual patch in the whole image. Each of these embeddings are fed into a transformer encoder. As it is usual in CNNs and RNNs, the model architecture finishes with a fully connected layer for label assignation and image classification. Figure 3.2 provides a visual explanation for the ViT framework. Xu, Y. et al. [37] built a model to infer human pose keypoint coordinates with a ViT as a backbone. This vision transformer extracts feature maps for the given persons. Feature maps are upsampled by a decoder to generate heat maps with respect to the joint keypoints. Since the model is trained with the MS COCO dataset, it is able to estimate 17 human pose keypoints. Figure 3.3 shows the performance of ViTPose models compared to other state of the art models.

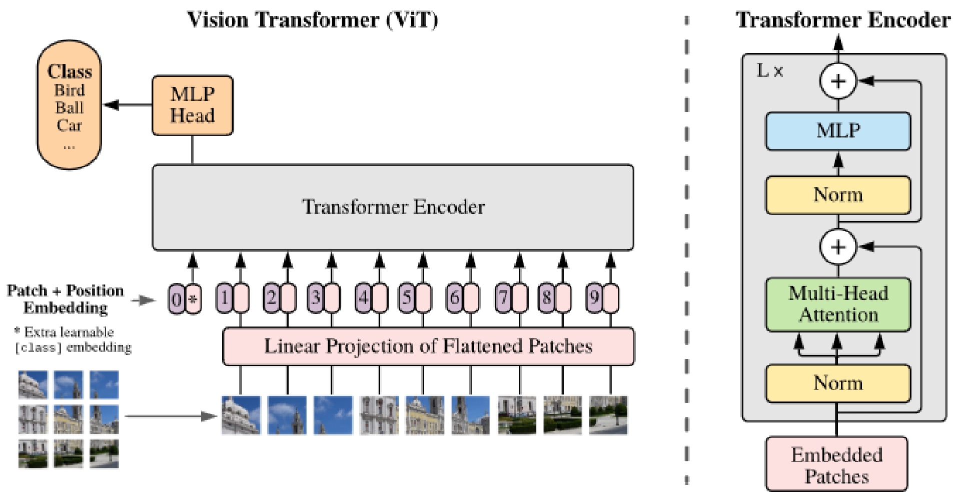


Figure 3.2: General overview of the ViT framework (left section) and the block diagram of a transformer encoder (right section). Figure extracted from [56].

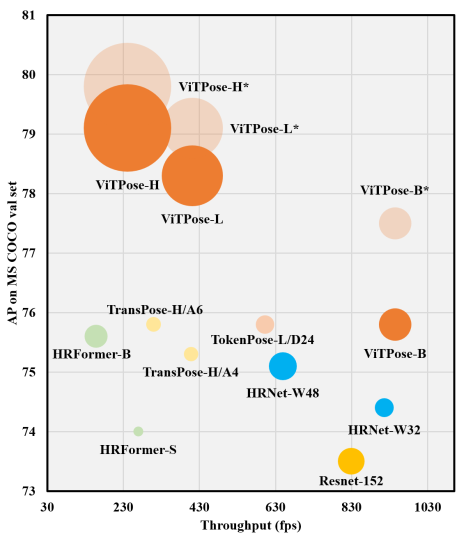
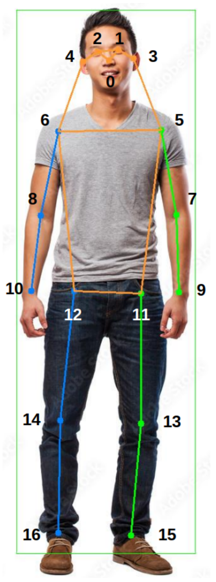


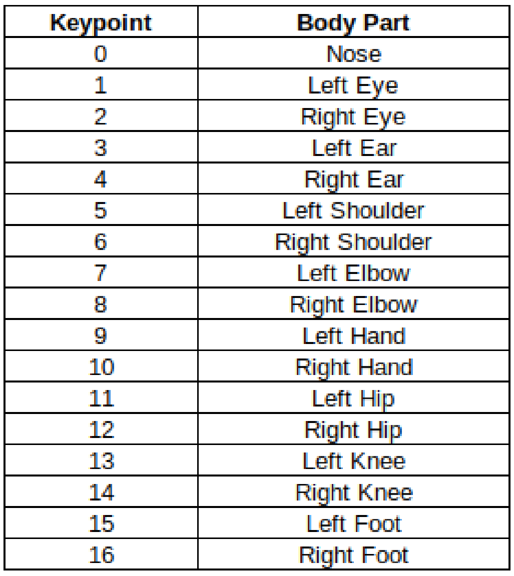
Figure 3.3: Comparison of ViTPose models against several state-of-the-art methods on the MS COCO dataset. Figure extracted from [37].

As it can be observed, the comparison showcases the throughput (in frames per second), the model size (in number of model parameters) and precision (in average precision for the validation set) for different pose estimation models. The size of the bubble in the graph is an indicator of the number of parameters it contains. ViTPose-B, despite being the smallest ViTPose model, is able to reach state of the art results in the MS COCO dataset. It is also the fastest model with that level of precision. ViTPose-L and ViTPose-H, albeit slower, are able to reach even higher levels of precision. The versions with an * beside their name are the result obtained by the same model when trained on multiple datasets.

While ViTPose itself only estimates the keypoints for the main body, this pose estimator can be coupled with different algorithms and models to also provide hand keypoint estimation. Different combinations will be tested for this project, to gauge which provides better results. Figure 3.4 shows the main keypoints estimated by this model.

ViTPose uses a *top-down* setting for human pose estimation. This means it first requires a detector to identify a human in the image before estimating the keypoints on it. To perform the detection task, MMDetection [36] open source object detection toolbox will be used. Similarly, the MMPose API [35] will be used to load the ViTPose model and combine it with another hand pose estimation model. Results of this detection and pose estimation implementation can be found in Chapter 4.





Keypoint	Body Part
0	Nose
1	Left Eye
2	Right Eye
3	Left Ear
4	Right Ear
5	Left Shoulder
6	Right Shoulder
7	Left Elbow
8	Right Elbow
9	Left Hand
10	Right Hand
11	Left Hip
12	Right Hip
13	Left Knee
14	Right Knee
15	Left Foot
16	Right Foot

Figure 3.4: Reference for the joint keypoints estimated by ViTPose.

3.2 Pose Augmentation

As previously mentioned, pose augmentation will provide an improvement in robustness for the depth estimations and the dynamic gesture recognition overall. This augmentation will be used to exploit the relative distance between joints, the relative orientation between limbs and the relative orientation of the limbs between the main body. As observed on [57], the use of these articulated pose features strongly improves the results in gesture recognition models. Embedded information in the augmented pose vector explains better the specific position of an individual when compared to only using a small set of joint keypoint coordinates. The augmented pose vector will use the estimated pose as a starting point and will be constructed in the following way:

1. A vertical line of best fit to the y position for all keypoints is calculated with the initial joint coordinates. An example of the line of best fit can be observed in Figure 4.6.This calculation is performed via least squares. The a and b parameters from $y = a * x + b$ that describe the line of best fit are included into the augmented pose vector.

2. Defining $\mathbf{J}_i$ and $\mathbf{J}_j$ as the i-th and j-th joints from the estimated 17 pose keypoints, vector $\mathbf{V}_{i,j}$ will be computed. This includes connected joints and non-connected joints. This vector will be calculated for all joints in the initial estimated pose and it is non-repeating. That is, $\mathbf{V}_{1,2} = \mathbf{V}_{2,1}$, and only the first one will be calculated.

$$\mathbf{V}_{i,j} = \mathbf{J}_i - \mathbf{J}_j \tag{3.1}$$

3. The length for each vector calculated in step 2 is computed and included into the augmented pose vector.

4. For each pair of anatomically connected joint vector, an angle is calculated between them.

5. For each vector, including the ones previously stated, an angle is calculated between the vector and the line of best fit.

This augmented pose vector is used for the estimation of the depth of the body. A similar procedure is implemented for the augmentation of each estimated hand pose. In this case, the augmentation will be used for the estimation of the depth of the left and right hand.

3.3 Depth Estimator

As previously mentioned, obtaining the key depths of the body will help to normalize the pose skeleton. Since absolute depth and scale information is lost when using monocular images, as in our case, it is proposed to use a learning-based depth estimator to approximate the depth of the neck keypoint and use it as a reference for the whole body. This estimator is a 9 layers neural network and will receive as input the augmented body pose keypoints when estimating the body depth.

3.3.1 Architecture

Since the depth estimation task will be based on a relatively small numerical input, and is only expected to deliver one numerical output, there is no need to build a deep neural network for this task. A small *n*-layers deep Multi-Layer Perceptron (MLP) will be used for the depth estimator. The general structure for an MLP is shown in Figure 3.5.

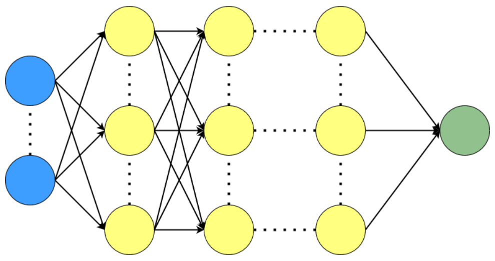


Figure 3.5: General diagram for a neural network with *n* fully connected layers.

A quick test different number of layers and different number of neurons per layer will be executed to determine the best model. The goal is to use the smallest possible model, while obtaining decent results. To determine the performance of these models, the error between the ground truth depth and the estimated depth will be used.

3.3.2 Training

When estimating the depth of the body, one main keypoint is selected from the pose estimation model. The chosen depth for ground truth, and for posterior estimation, for the main body is the approximated depth at the neck of the body. Pre-processing will be required for any dataset before training can be undertaken with the model. For each image in each RGB-Depth pair of the dataset, the pose will be estimated and augmented using the method outlined in section 3.2. Both the augmented pose and the extracted depth points will be used as additional inputs for training.

Several datasets are proposed in Table 3.1 for the training of the learning-based depth estimator. The NTU group of datasets provide data with a great resolution, they contain the biggest number of different subjects, and provide a big number of samples. As such, NTU RGB+D was selected to perform the training of the model. This was selected in lieu of the 120 version to keep the data size manageable.

Datasets for Learning-Based Depth Estimator		
Dataset	Size of Dataset	Data Available
NTU RGB+D [58]	60 action classes 56,880 video samples 40 subjects	1920x1080 RGB videos 512x424 depth maps 512x424 IR videos 25 body joints
NTU RGB+D 120 [59]	120 action classes 114,480 video samples 106 subjects	1920x1080 RGB videos 512x424 depth maps 512x424 IR videos 25 body joints
Isolated Gesture Recognition (ICPR '16) [60]	249 gesture classes 47,933 video samples 21 subjects	320x240 RGB videos 320x240 depth maps
OpenSign [61]	10 static classes 10 volunteers	.png images .bin depth images

Table 3.1: Datasets for training the depth estimators. Key characteristics for these datasets are variability in images and access to the depth maps. Access links for these datasets can be found in the references.

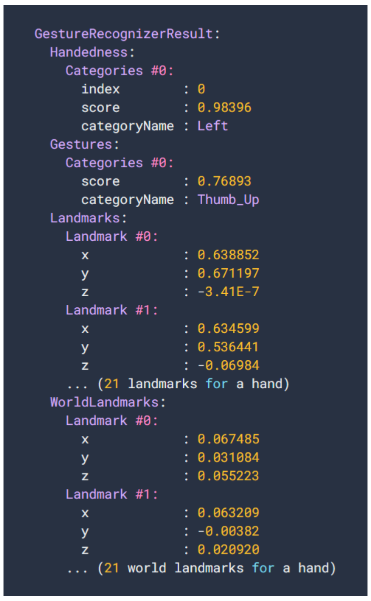
3.4 Focus on Hands

The goal of the Focus on Hands module is to locate the hand of the diver and to crop the image in such a way that it only contains the hand and immediate surroundings. This process is divided in two parts: hand localization and dimensioning of the bounding box. Hand localization is performed by running the MediaPipe hand landmarker model on the whole image. This model outputs an instance of hand information for each hand detected in the image. This instance contains, among other information, the pixel coordinates of each hand joint identified, the confidence of it being correctly identified, and an estimation of the proper body side location; to make it clearer, the model labels each hand structure as part of the left or right section of the body. A summary of the estimated features for each detected instance can be observed in Figure 3.6.

By making use of MediaPipe's settings, we can limit the maximum number of hands to be located on each frame to two. This helps to reduce the probability of misidentification and handling more data than expected.

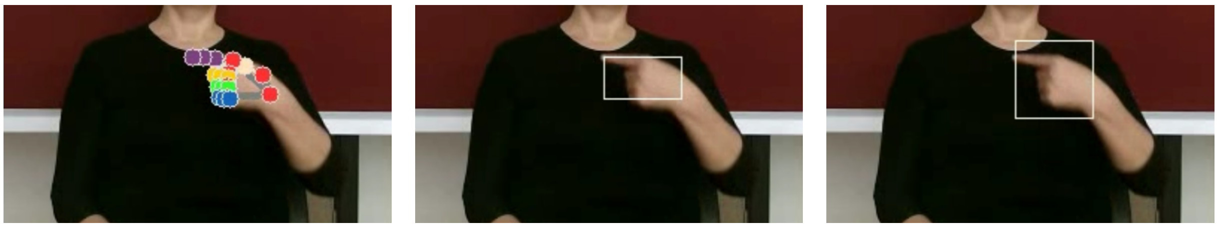
For each identified hand in every frame, the landmark coordinates are extracted and processed to select the minimum and maximum values for each coordinate. This procedure is basically extracting the corners for the rectangle in which the hand is located. However, a couple of corrections and checks need to be performed to ensure a quality hand extraction with this method:

1. **Bounding Box Resizing.** One of the key requirements for the CNN model used to process the hands is the image size. A square image input size needs to be used. This is solved before hand cropping to avoid stretching the image before sending it to the CNN model, which could add unwanted noise to the process. Once the hand poses are estimated, the algorithm checks for the larger side. The smaller side coordinates are extended to match its length, as seen in Figure 3.7.



```
GestureRecognizerResult:
  Handedness:
    Categories #0:
      index        : 0
      score        : 0.98396
      categoryName : Left
  Gestures:
    Categories #0:
      score        : 0.76893
      categoryName : Thumb_Up
  Landmarks:
    Landmark #0:
      x            : 0.638852
      y            : 0.671197
      z            : -3.41E-7
    Landmark #1:
      x            : 0.634599
      y            : 0.536441
      z            : -0.06984
    ... (21 landmarks for a hand)
  WorldLandmarks:
    Landmark #0:
      x            : 0.067485
      y            : 0.031084
      z            : 0.055223
    Landmark #1:
      x            : 0.063209
      y            : -0.00382
      z            : 0.020920
    ... (21 world landmarks for a hand)
```

Figure 3.6: The model is able to identify the handedness of the hand, the landmark coordinates and world landmark coordinates. The landmark coordinates are normalized to a [0,1] range, using the image width and height. The world landmark coordinates units are meters, using the hand's geometric center as origin.



(a) Hand keypoints estimation. (b) Raw hand bounding box. (c) Resized hand bounding box.

Figure 3.7: Hand bounding box resizing process on a ChaLearn LSSI Dataset example.

2. **Hand Temporal Closeness Check.** One of the disadvantages of using a model to estimate the side of each hand is the possibility of misidentification. Initial tests performed with MediaPipe showed that this is not a frequent issue, but it does occur from time to time, especially when the hands perform a sudden movement. To reduce the impact of this issue, a check is performed to make sure that this estimated side is not swapped in consecutive frames. This is done by comparing the center of the hand in consecutive frames. If the difference is below a threshold of 10%, the estimation is accepted as correct. If not, this is solved according to the subsequent item.

3. **Missing Coordinates Replacement.** Coordinates have to be replaced if a hand was not detected, or if it was misdetected, in the current frame. This replacement is performed within a window of 3 frames. The first step is to calculate a delta between frames $t-2$ and $t-1$ for all coordinates. Once this is done, the delta is added to the coordinates of frame $t-1$ to get an estimation of the current coordinates. This replacement cannot be done more than three frames in a row, as the hand may have moved well beyond the estimation by this point.

These steps allow a smooth hand cropping throughout a video. The goal of the Focus on Hands module, can be seen in Figure 3.8.

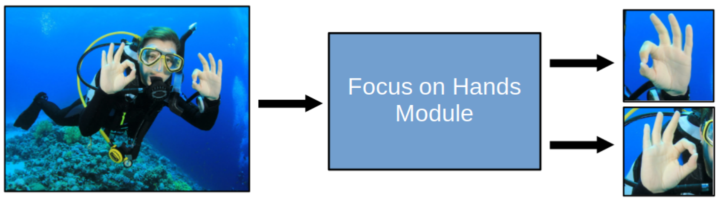


Figure 3.8: The FoH module crops two sections from the main image, centered on each hand.

3.5 Pose Pre-Processing

As previously stated, the Pose Pre-Processing is divided in three steps: skeleton filter, skeleton position and scale normalization, and skeleton depth estimation. Since depth estimation has already been explained, this section will provide an explanation for filtering and normalization.

3.5.1 Skeleton Filter

The goal of the skeleton filter is to reduce occasional jitter in the estimated keypoints, minimize missing joint coordinates between successive frames that may be detrimental in the training and estimation phases for the model. This filter works on a window of N consecutive images, whose size is adjustable and results will be evaluated with N values of 5, 7, and 9. Defining $J_n^i = (x_i, y_i)$ as the coordinates of the i^{th} joint in the n^{th} image of the window.

First, missing joint coordinates are replaced with the coordinates of the same joint in the previous frame. This replacement must meet three requirements to take place: (a) J_n^i has not been detected, (b) it is existant in all previous images of the window, and (c) it has not been consecutively replaced N times. In a second step, Gaussian smoothing is applied to each J^i over the N images of the window.

Figure 3.9 exemplifies the skeleton filter implementation for 7 consecutive images.

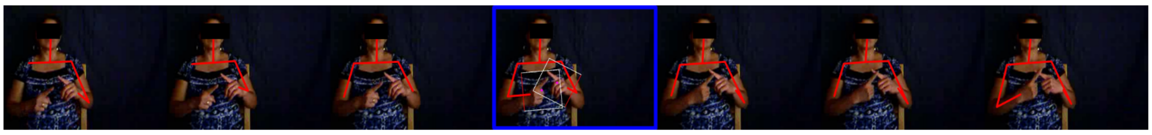


Figure 3.9: Example of the skeleton filter with a window of size 7. The image outlined in blue shows the result from the filter, while the other 6 are the result of pose estimation. Even though adjacent images are lacking keypoints, the focused frame has correctly estimated their position. Image extracted from [49].

3.5.2 Skeleton Normalization

The purpose of this step is to standardize the position and the size of the estimated pose keypoints. Position normalization is performed by substracting the average of the coordinates from both shoulder keypoints to each joint, J^i. This eliminates the influence of the user's location in the frame. Scale normalization is performed by dividing the position normalized joint coordinates by the depth of the body.

3.5.3 Pose Re-Augmentation

Once the pose keypoints have been normalized, pose augmentation is performed again to construct a normalized augmented pose vector. This augmented information will be used as input for the neural network architecture to identify dynamic gestures.

The process for the augmentation of the normalized pose follows the same steps explained in section 3.2.

3.6 Neural Network Architecture

The proposed solution to the underwater gesture recognition problem is the use of a neural network architecture designed for the identification of static and dynamic gestures. These gestures will be later mapped to primitive motions for the vessel. The baseline for the proposed neural network architectureis StaDNet and can be observed in Figure 3.10.

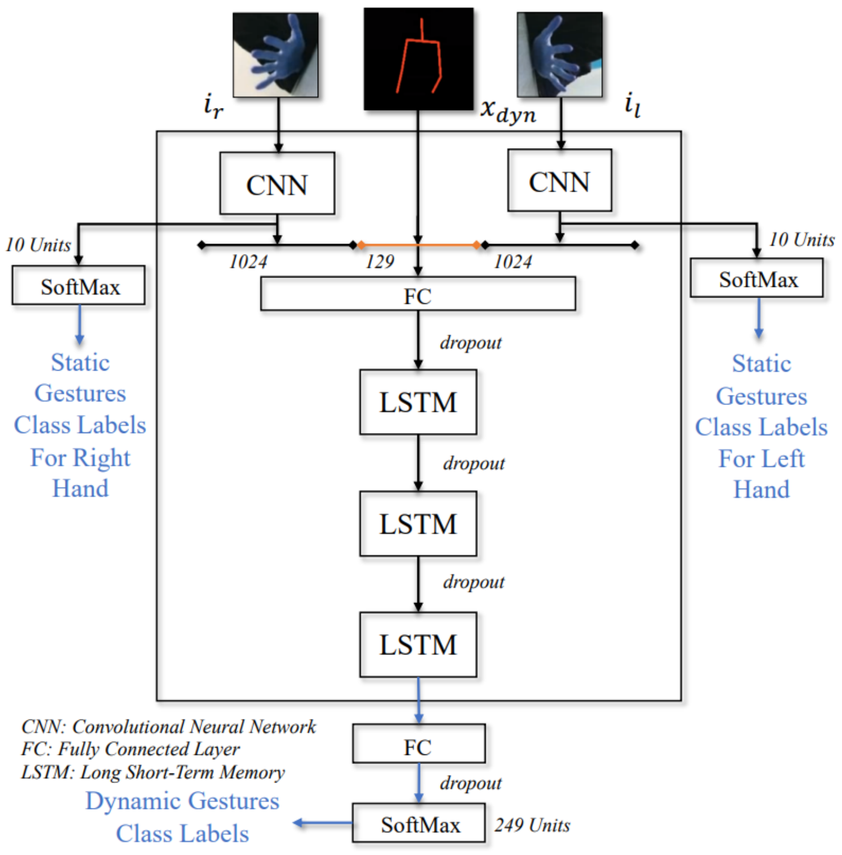

i_r
x_{dyn}
i_l
CNN
CNN
10 Units
10 Units
1024
129
1024
SoftMax
SoftMax
FC
Static Gestures Class Labels For Right Hand
Static Gestures Class Labels For Left Hand
dropout
LSTM
dropout
LSTM
dropout
LSTM
CNN: Convolutional Neural Network
FC: Fully Connected Layer
LSTM: Long Short-Term Memory
FC
dropout
Dynamic Gestures Class Labels
SoftMax
249 Units

Figure 3.10: Illustration of StaDNet. Figure extracted from [49].

On this baseline, the identification of static gestures is performed by a CNN that processes two independent images, cropped and focused on each of the diver's hands. The CNN is a modified Inception v3 [62] model to output an embedding of size 1,024 that contains rich information on the hand image. A brief diagram on the architecture of Inception v3 can be observed in Figure 3.11. This embedding is fed into a fully-connected layer with a softmax function to classify a static gesture. The recognition of static gestures depends heavily on the correct extraction of the input image and the training performed on this CNN model. Inception v3 requires an input of size 299x299x3, which works well with the cropped sections for the static hand gesture estimator. A resizing step will be implemented in the framework to ensure the correct input size for each of the cropped hand images. On this section of the architecture, the main difference with the baseline implementation is the dataset used for finetuning. The baseline uses an in-house developed dataset for their training, which is currently unavailable. Whereas we use a publicly available hand gesture dataset, 27 Class Sign Language Dataset [63], in our implementation. This dataset features 173 different individuals performing gestures and each image has a resolution of

3024x3024 pixels.

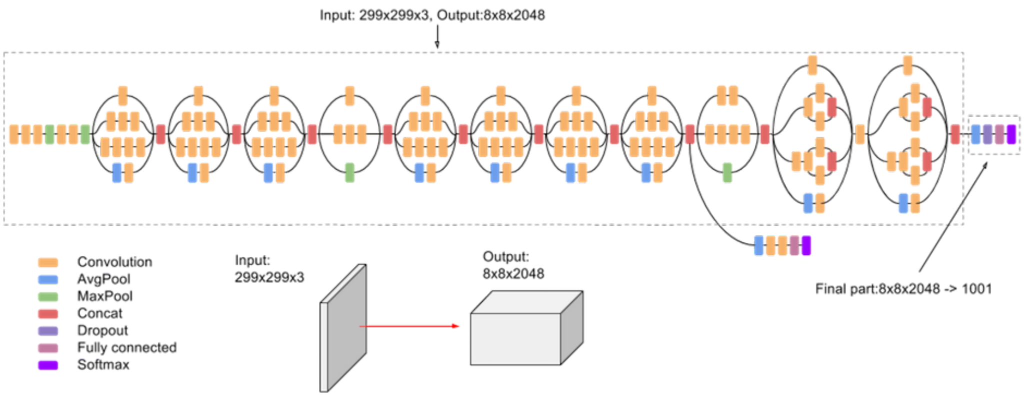


Figure 3.11: Inception v3 model's high-level diagram.

The identification model for dynamic gestures receives as input a fused embedding vector for each of the hands with an augmented pose vector. This pose vector is the result of applying a human pose estimator to the input image to estimate the position of a basic skeleton and generating mathematical relationships between the keypoints of this skeleton. The fused vector is then processed by a fully connected layer and several LSTM blocks. A reminder of a basic LSTM block is detailed in Figure 3.12. Dropout is performed between these layers to prevent over-fitting and obtain a better generalization. Finally, and similarly to the end of the static gesture detector, the network finishes with a fully connected layer and a Softmax function to classify the dynamic gesture.

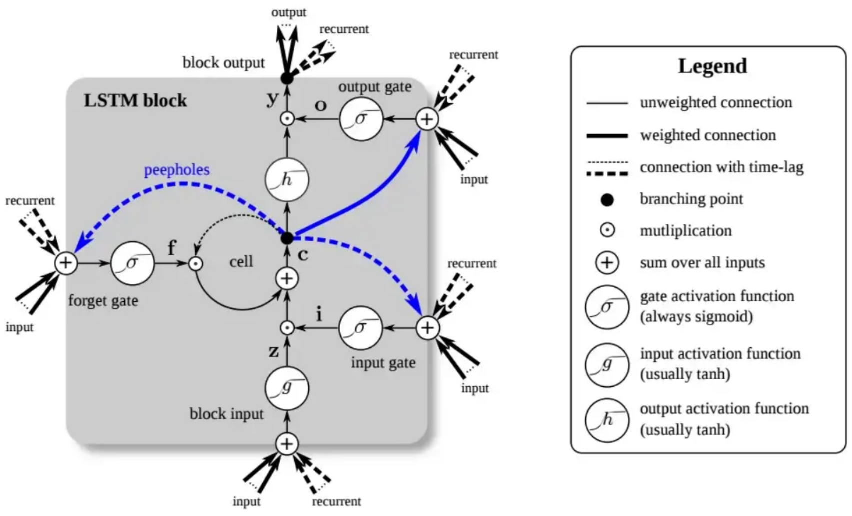


Figure 3.12: The LSTM block has four input weights as well as four recurrent weights. Peepholes are additional connections between memory cells and gates, but they do not significantly improve performance and are frequently eliminated for simplicity. Representation created by Greff K. et al., which was published in LSTM: A Search Space Odyssey [64].

3.7 Model Training

The training for the dynamic gesture recognition model is performed in two phases. The first training phase uses the ChaLearn LSSI Dataset (CVPR'21). This dataset features 226 signs performed by 43 different signers. It contains 20 different backgrounds and has people both standing and sitting. This dataset was chosen to increase the variability in the samples and to improve the input quality to the model. When compared to the dataset used for the original StaDNet implementation (ChaLearn Continuous Gesture Dataset (ICPR'16) [65]), this dataset contains double the number of signers and has a higher video resolution.

To feed the dataset to the model, each video has to be processed with the procedures outlined in sections 3.1 - Robust Human Pose Estimation, 3.2 - Pose Augmentation, 3.4 - Focus on Hands, and 3.5 - Pose Pre-Processing. To avoid performing this process for each video on every training epoch, it was decided to run the Spatial Attention Module algorithm beforehand and save the results for fast access during training. The conversion of video to a list of augmented fused vectors can be observed in Figure 3.13.

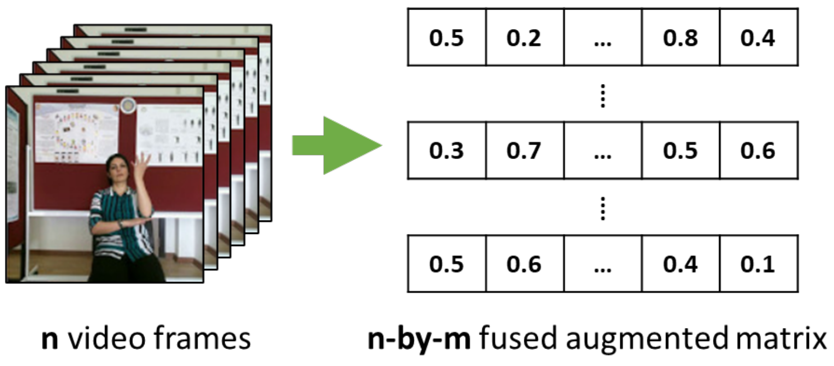
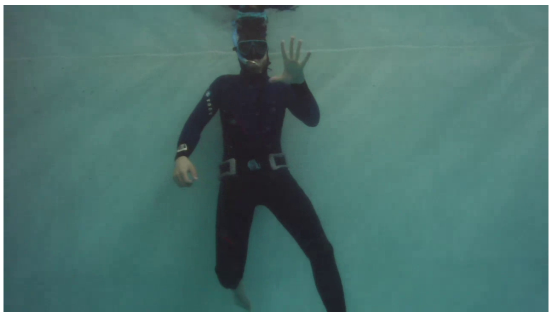


Figure 3.13: Each video of n frames is processed and converted into a list of n augmented fused vectors.

On the second training phase, the model is finetuned with a small dataset of underwater gestures of our making. This dataset is detailed in section 3.8. The same pre-processing approach is adopted for the finetuning of the model.

3.8 Underwater Dataset Recording

Since no adequately labeled underwater dynamic gesture datasets were found for the training of the model, it was decided to record a small dataset for preliminary training and testing. Twenty gestures were chosen from common diver signs. This list of gestures contains examples related to motion commands and information sharing. The complete list of gestures and their meaning is detailed in the mosaic included in annex A. Two examples of the aqcuired samples in this dataset can be observed in Figure 3.14.


(a) Subject A performing a gesture underwater.

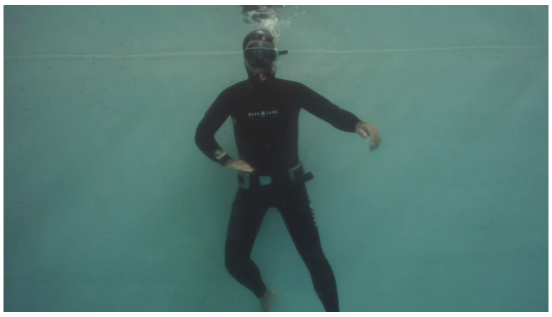
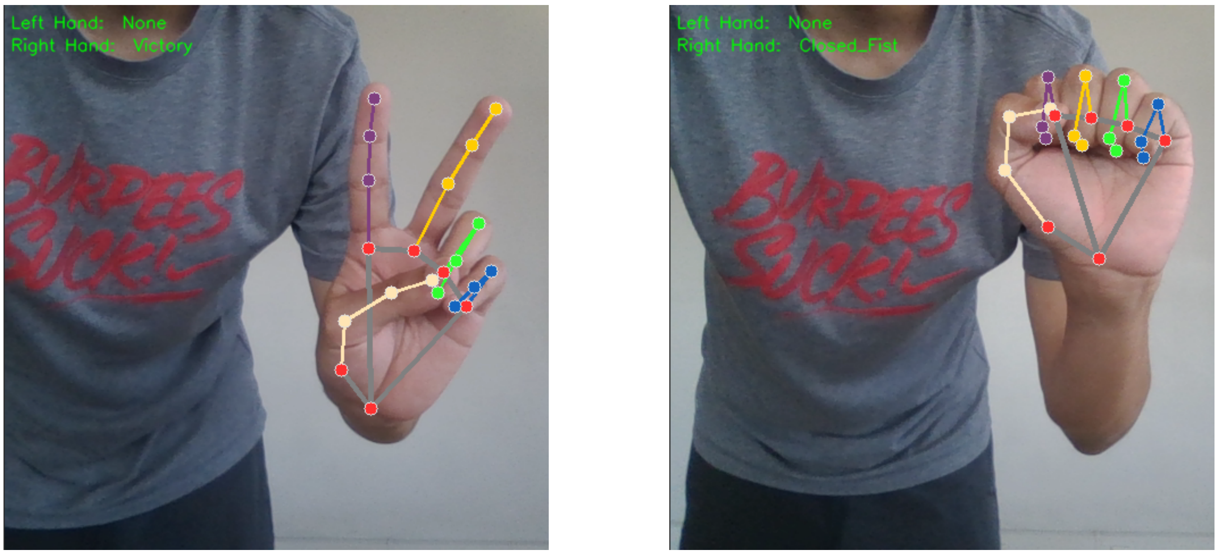

(b) Subject B performing a gesture underwater.

Figure 3.14: Snapshots of subjects performing diver gestures underwater. This was recorded by the 1080p camera of a BlueROV2 Heavy.

3.9 Gesture Identification

Once the model is fully trained on the underwater gestures dataset, a live testing demo is built, which is based on the approach mentioned in section 2.6.2. The MediaPipe static gesture recognition model will be used to detect certain key gestures. To begin recording, the user will need to perform 'V' signs with both hands. To end the recording, the user will have to close both fists in front of the camera. The begin and end gestures can be observed in figure 3.15.



(a) 'Victory' sign. (b) 'Closed Fist' sign.

Figure 3.15: Start and end gestures detected by the MediaPipe Hand Gesture Classification model. These gestures will have to be performed with both hands for a couple of successive frames to successfully trigger an event.

Once the recording is finished, a number of frames equivalent to 2 seconds are removed at the ends to eliminate any motion related to the static gestures. A visual representation of the timeline of how this works is presented in figure 3.16

Afterwards, the frameworks performs the complete processing of the cropped video. The recognized gesture will be printed in the terminal, along with its estimated probability. The purpose of this demo is to have a live demonstration that can be shared with others to explain the process behind the framework. Once the gesture detection algorithm is ensured to work satisfactorily, it can trigger the underwater robot motion, alarm signal, or any other task, instead of merely displaying the identified gesture on the screen.

3.10 Gesture to Motion Mapping & AUV Motion Control

To map the gesture recognition system, and the execution of motion primitives, a Finite State Machine (FSM) will be implemented to track dynamic gesture states. After a correct gesture detection, the FSM will change to the execution state, where it will perform the selected motion primitives described in Section 2.8. As the gesture detection is performed offline, another gesture can be input to the framework while the vehicle is performing a task.

AUV Motion Control will be done using ROS and the high-level vehicle operation software stack Free Autonomous Robots for Observations and Labeling (FAROL) [66] available as part of the Dynamical Systems and Ocean Robotics Lab (DSOR), part of the Institute of Systems and Robotics | Lisboa (ISR|Lisboa). This stack has been used for multiple path following experiments with the Medusa marine

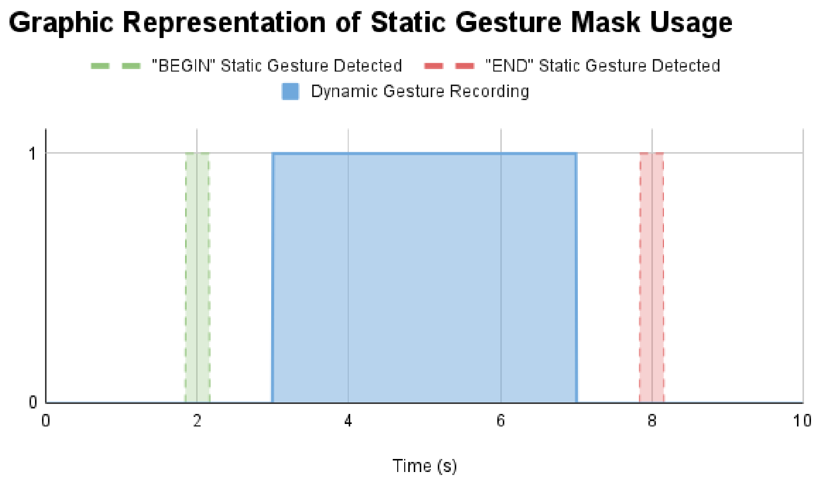


Figure 3.16: Timeline of the gesture detection and recording for a 10s video. In this example, the static gestures to begin are detected at 2s, while the static gestures to finish are detected at 8s. When the begin gesture is detected, a number of frames are skipped before starting to record. After the end gesture is detected, the last frames of the recorded video are removed. Thus, the duration of the recorded video is smaller than the time between both static gesture detections.

vehicle, as shown in [67].

Chapter 4

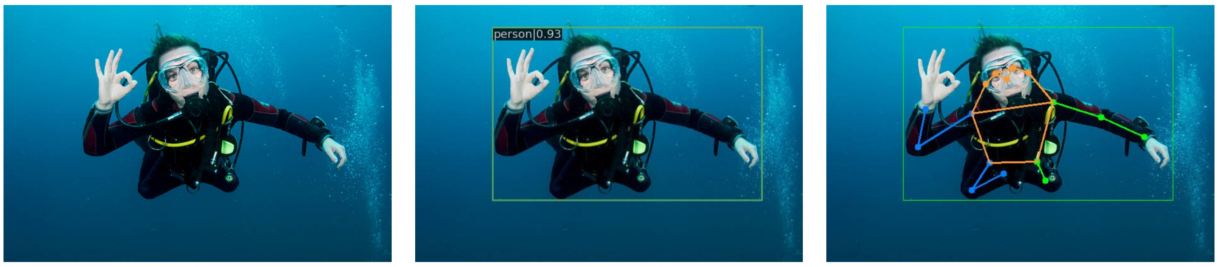
Implementation Results

On this chapter, the results from the implementation of the models and from the performed tests will be presented. The chapter will be divided accordingly to present and discuss performance from the following topics:

- Robust Human Pose Estimation;
- Pose Augmentation;
- Hand Extraction;
- Dataset Pre-Processing;
- Model Training;
- Underwater Dataset Recording;
- Live Demo;
- Live Simulation.

4.1 Robust Human Pose Estimation

An implementation of ViTPose, using the MMDetection and MMPose APIs as base, has been used for human pose estimation. A module was developed that takes an image as an input and returns the pose keypoints information along with the estimated human skeleton on top of the original image. Example results for human detection and pose estimation can be observed in figure 4.1. It is important to note that the detection model and the pose estimation model have only been trained on “dry land” datasets. None of these models have been trained using underwater images. As such, there are some instances where either the detection or the pose estimation does not work as expected. Nevertheless, these issues only come up on instances of extreme occlusion or very low visibility.



(a) Input image to perform human pose estimation on. (b) Bounding box detected from the input. (c) Pose keypoints estimated with ViTPose.

Figure 4.1: Preliminary results for the two steps to perform human pose estimation.

As can be observed, human detection performs well underwater, at least for favorable lightning conditions. Even if the whole body is not in view, and is not in the typical upright position, the detector is still able to identify the location of the human in the image. Both the detector and the estimator provide some measure of confidence on their inferences. In some cases, the pose estimator is only confident enough on the location of some of the keypoints of the body. In these situations, where joints are below the confidence threshold, these pose keypoints would usually be excluded and removed from further steps. This issue will be addressed as was proposed in chapter 3.5, exploiting the continuity of previous frames.

Even though the detection and pose estimation models provide good results in underwater scenes, some degree of noise can be noticed in continuous frames pixel coordinates. Nonetheless, the implementation of the algorithm proposed in 3.5 does not only replace missing keypoints, it also smooths the variation of pixel coordinates over time to reduce the noisy estimation. Figure 4.2 shows the untreated estimated pose keypoints of a diver’s wrists throughout the “Low on Air” dynamic gesture. It is of note how the motion is jagged and it has a couple of outliers from an ill estimated frame. This poses a threat to the training of the dynamic gesture recognition model, as we cannot expect the noise to behave regularly in similar motions. If untreated, it could cause the optimizer to focus on the wrong features of the dynamic gesture. Another issue with the raw pose estimation is the missing keypoints along the motion. This particular video has 8% of the frames with one or more missing keypoints, 11 out of 137 frames. This gap in information could make it hard for the model to properly learn the different gestures, specially in a small dataset.

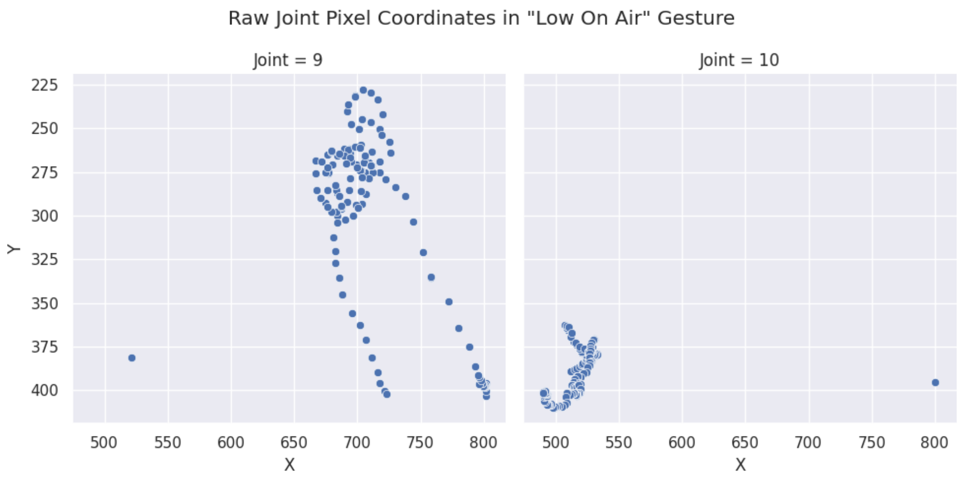


Figure 4.2: Unprocessed pixel coordinates of the left wrist (joint 9) and right wrist (joint 10) keypoints of a diver performing the **Low on Air** dynamic gesture. This gesture is performed by placing a closed fist in front of your chest and rotating it several times.

Now, with the implementation of the pose pre-processing algorithm, an improvement on the pixel coordinates can be observed in Figure 4.3. The jaggedness of the motion is reduced, missing keypoints have been replaced and the outliers have been removed.

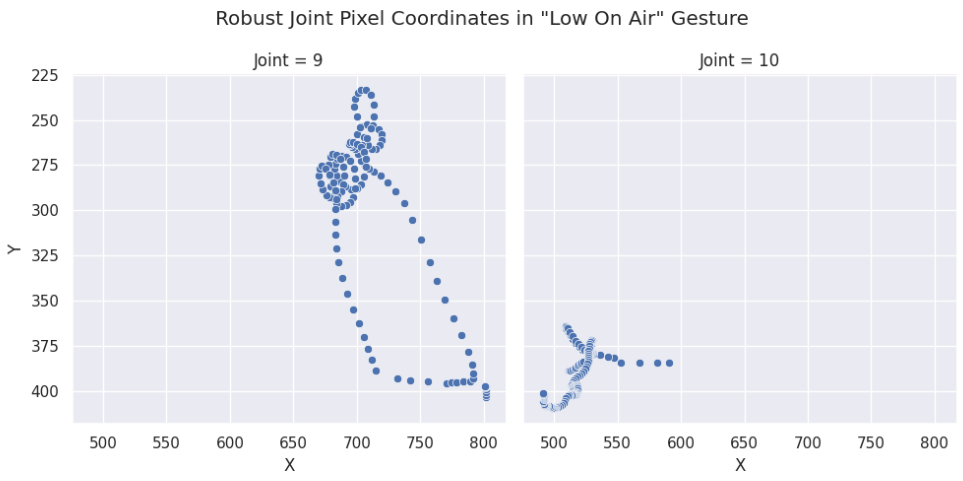


Figure 4.3: Processed pixel coordinates of the left wrist (joint 9) and right wrist (joint 10) keypoints of a diver performing the **Low on Air** dynamic gesture. The coordinates are less noisy and it's easier to follow motion patterns.

Two more comparisons are shown in Figures 4.4 and 4.5. The scenarios in these comparisons are labeled as *UnderwaterGesture02* and *UnderwaterGesture03*, which are two different gestures being performed by a diver. In the *UnderwaterGesture02* scenario, the diver performs the **Look Up** gesture, while in the *UnderwaterGesture03* scenario, the diver performs the **Slow Down** gesture. The motion for these gestures can be observed in Annex A.

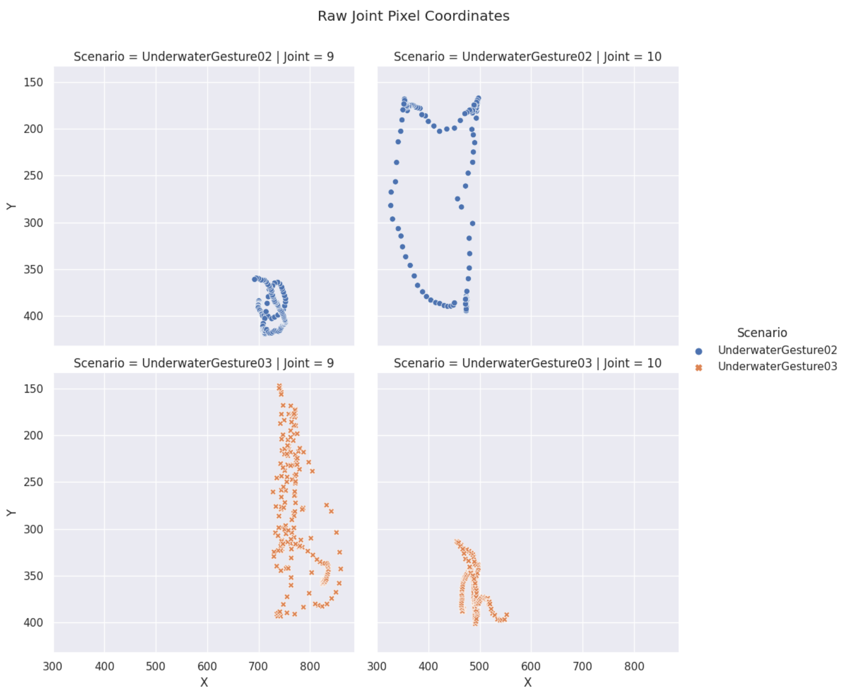


Figure 4.4: Unprocessed pixel coordinates of the left wrist (joint 9) and right wrist (joint 10) keypoints for two different scenarios. In scenario *UnderwaterGesture02*, a diver is performing a **Look Up** gesture. While in scenario *UnderwaterGesture03*, it signs **Slow Down**.

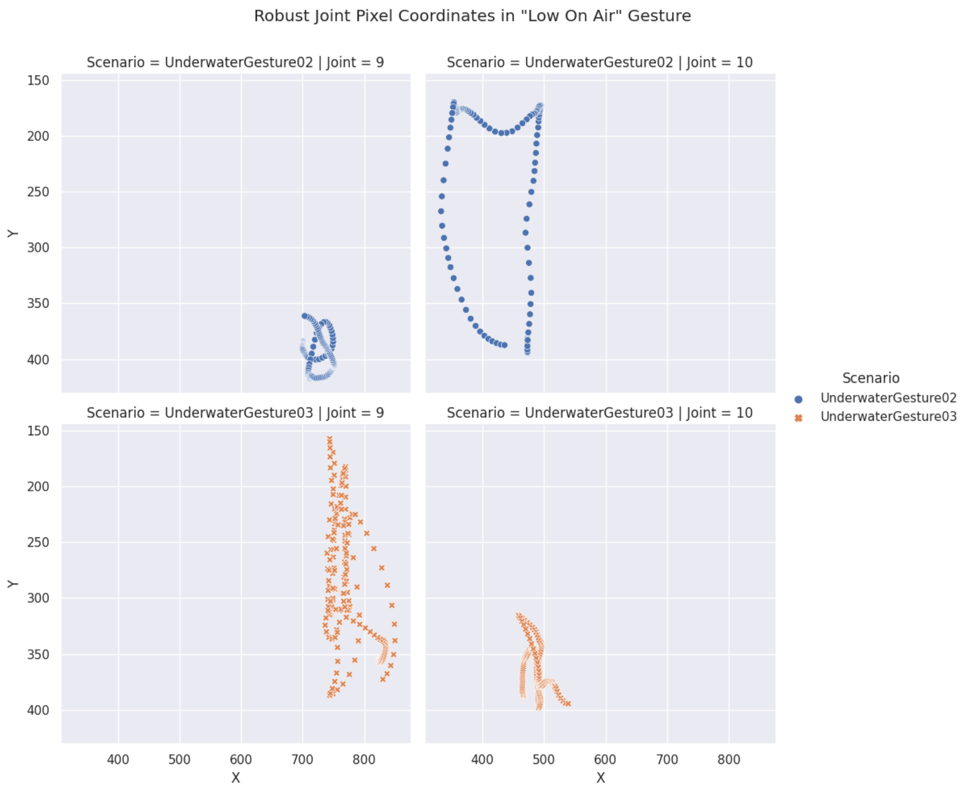


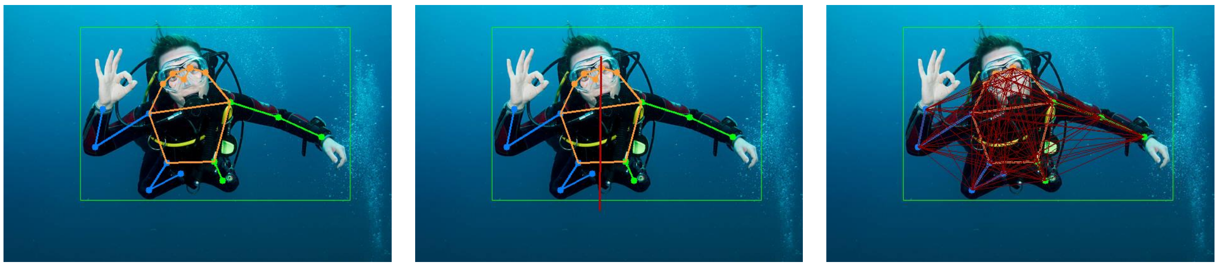
Figure 4.5: Processed pixel coordinates of the left wrist (joint 9) and right wrist (joint 10) keypoints for two different scenarios. The gestures are the same as shown in Figure 4.4. As observed in the previous comparison of Figures 4.2 and 4.3, the motion of the wrists is also better defined and easier to follow on these cases.

4.2 Results for Pose Augmentation

Pose augmentation is performed using the robust keypoint information extracted from the previous step. As mentioned in Chapter 3, the goal of this process is to create a vector with information containing the relative position and orientation of the limbs in the human body with respect to each other. It is important to note that pose augmentation is only performed on the information of joints over the confidence threshold. The value of this confidence threshold was selected empirically, through trial and error, to minimize the number of wrong inferences without cropping too many correct estimations.

The first information included in the augmented pose vector is the coordinates of the filtered estimated keypoints. Using these filtered coordinates, the a and b parameters that describe the line of best fit are calculated using the least squares method. This $[a,\, b]$ pair is appended to the augmented pose vector. Then, for every estimated joint, the relative distance in x and y is calculated to every other joint. As the absolute value of the distances between joints J^i and J^j is equal to the distance between joints J^j and J^i, only one is calculated. After all the vectors have been calculated, their length is also calculated an appended to the augmented pose vector.

Initial results on the two first steps of pose augmentation can be observed on Figure 4.6.



(a) Estimated pose. (b) Line of best fit. (c) Joint-to-joint vectors.

Figure 4.6: (a) Pose augmentation begins with the joint keypoint coordinates from the previous step. (b) A line of best fit is calculated for the whole estimated pose. Parameters *a* and *b* are saved into the augmented pose vector. (c) Vectors are calculated between each anatomically and non-anatomically connected joint keypoints. These are non-repeating.

Finally, angle calculation is performed and included in the augmented pose vector. First, the angles between each pair of connected vectors are calculated. This includes, for example, the angle between the vector from the hand to the elbow and the vector formed between the keypoints for the elbow to the shoulder. This step does not include vectors that are not anatomically connected. Lastly, for every calculated vector, an angle is calculated using the line of best fit as a reference. While these angles are not visualized, they are key to extracting useful features for our model. The baseline implementation has an augmented pose vector of size [1, 129], whereas our implementation of this augmented pose vector is of size [1, 303].

The contents of the baseline implementation of the augmented pose vector are categorized as follows:

- 42 elements from abscissas and ordinates of the augmented vectors;
- 21 estimated lengths of the augmented vectors;
- 34 relevant angles;
- 16 joint velocity vectors;
- 16 joint acceleration vectors.

The contents of our implementation of the augmented pose vector are categorized as follows:

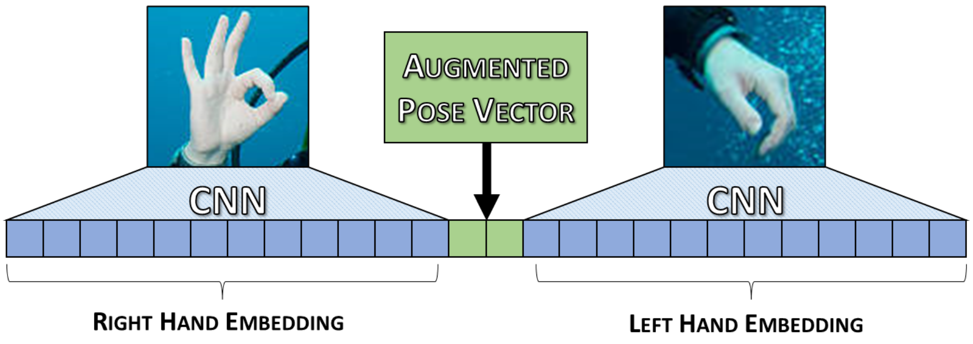


Figure 4.7: The fused vector is used as input for the dynamic gesture recognition model and is composed of the right hand embeddings, the augmented pose vector, and the left hand embedding. The total vector size is [1, 4399].

- 22 robust human pose keypoints coordinates;
- 110 elements from abscissas and ordinates of the augmented vectors;
- 125 relevant angles;
- 22 joint velocity vectors;
- 22 joint acceleration vectors;
- 2 elements describing the line of best fit.

Our implementation contains two additional categories besides those common elements in both implementations. The common categories are the augmented vectors, angles, velocity vectors, and acceleration vectors. In our implementation, we decided to include the information regarding the position of each joint and the line of best fit to these joints. The position of each joint is described as the position-and-scale normalized coordinates derived from the procedure shown in 3.5.2. The line of best fit gives us an additional idea of the limb movements with respect to the whole body.

4.3 Dataset Pre-Processing.

The output data of the augmented pose vector and the hand extraction for each frame is merged into one vector, as shown in Figure 4.7. It is important to remember that this data comes from frame by frame input fed sequentally to the Spatial Attention Module. This means that, if we were to train our neural network using the complete pipeline, all of the frames in each gesture video would need to be processed repeatedly on every epoch of the training period. In a scenario where we train our neural network for 100 epochs, every frame in every gesture video would need to be processed 100 times. This would greatly increase the time needed to train our model, knowing that total training duration can go much higher than 100 epochs.

To address this issue we pre-process and save every gesture video in our datasets of interest. This process is quite straightforward using the methods we developed in the previous chapters, but it will be outlined in this paragraph. Once we have the gesture videos of the dataset in a single folder, we run through them one by one. Each frame of the video goes through the Spatial Attention Module, using the 7 frame window for robustness. The output of each frame is the fused vector shown in Figure 4.7. The vectors from a single video are then stacked on top of each other to create a larger data structure with the information of every video. This data structure is saved per video in numpy format once all of its frames have been processed. This format makes it fast and easy to load the data when needed during the training period.

The details of the datasets of interest for continuous gesture training are shown in Table 4.1.

Continuous Gestures Datasets

Attribute	ChaLearn LAP RGB-D [65]	ChaLearn LSSI [55]	Diver Gestures Dataset **(ours)**
Classes	249	226	20
Gestures Origin	Mixed Signs	Turkish Signs	Diver Gestures
Subjects	21	43	4
Video Samples	47,933	36,302	900
RGB Videos	✓	✓	✓
Depth Maps		✓	
Body Joints		✓	
Air Environment	✓	✓	✓
Underwater Environment			✓

Table 4.1: Characteristics of continuous gestures dataset.

4.4 Underwater Gestures Dataset Recording and Processing.

During the second week of July 2023, and supported by staff members from ISR-IST, a group of four volunteers were gathered to record underwater gestures in a water tank. A BlueROV2 Heavy, with its standard 1080p video camera, was used to record the people underwater. Since there is no widely available underwater gesture dataset, this was a required step to properly test the framework on a different scenario.

This scenario contained several key differences to above-water datasets. Mainly, the volunteers were using a full diving body suit and snorkel mask. This put to the test the performance of several key components in the framework. First, initial human detection models showed difficulties in correctly identifying people when wearing these suits. Second, pose estimation models have subpar performance when doing inference on such scenarios. Since the diving suit is usually completely black, it proves hard for estimators to correctly locate limbs and joints, specially in the lower body. This is mainly due to the fact that detector and estimator models are rarely trained with such attire.

Another key difference in our dataset is the underwater imaging characteristics. A light blue hue veils the complete image, creating a small distortion with respect to ordinary settings. The underwater setting may also carry the spontaneous appearance of bubbles in front of the face of the diver. This generates additional noise for the detectors.

The 20 proposed underwater gestures in section 3.8 were performed for this dataset, with 17 of them being one-handed gestures. This allowed us to increase variability by performing the gestures with each hand. Only 3 of the gestures did not allow variability by switching the hand: “I’m Cold”, “To the Right”, and “To the Left”. This results in 37 gestures to be performed in each round.

While the initial plan was to have each of the four volunteers perform four rounds, this was not possible due to time-constraints and a lack of experience with the medium. In the end, it was only possible to record two volunteers performing the gestures. However, each one of them performed a minimum of six rounds each. These recorded videos were divided and labeled into clips with one gesture each. The two

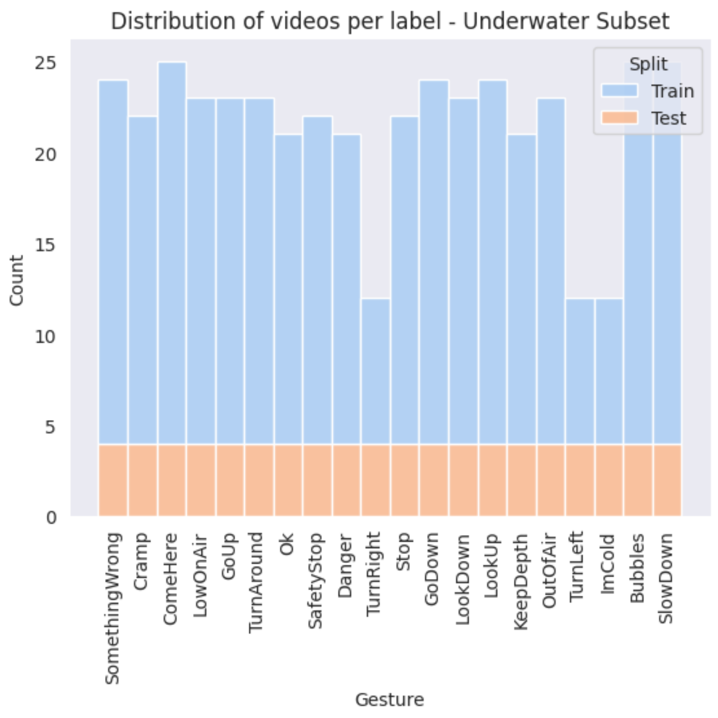


Figure 4.8: Most of the classes in our underwater dataset have around 22 video samples. Those with a lower video count are those previously mentioned without hand variability.

subjects that performed videos for this dataset are going to be referred to as subject A and subject B in the following chapters.

The final distribution of the gestures can be observed in Figure 4.8.

4.5 On Air Gestures Dataset Recording and Processing.

To increase the number of video samples, and to increase the variability in our dataset, the same gestures were performed on air by two additional subjects. These subjects are going to be referred to as subject C and subject D later on this work. The distribution of video samples per gesture can be observed in Figure 4.9.

4.6 Model Pre-Training

Before training our model with our dataset, a pre-training phase was carried out. This allowed us to take advantage of the larger continuous gesture datasets available publicly. Even if the gestures performed are not the same, this training will allow our model to learn generalizable parameters to be used as a stepping stone later on.

The pre-training of the model was carried out by using 47 classes of the ChaLearnLSSI dataset. A total of 5,350 video samples were used in total, along an 80/20 train/test split. The pre-training was carried out for a total of around 35,000 steps. The performance of the accuracy and the loss of the model can be observed in Figure 4.10.

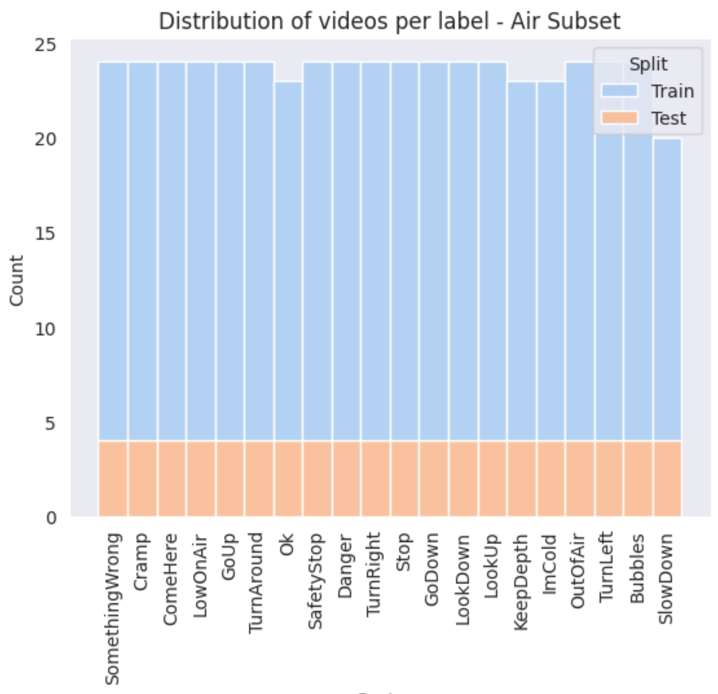
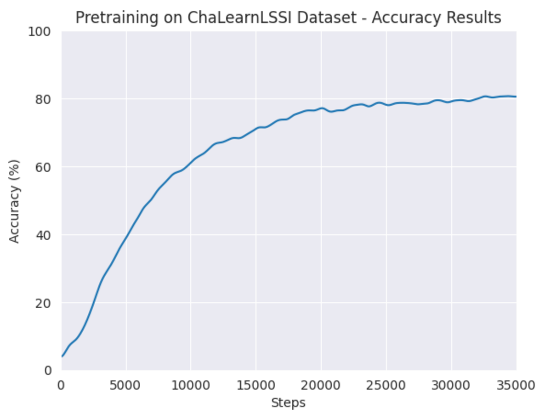
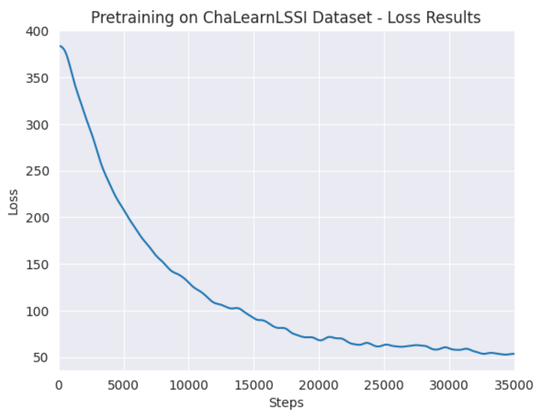


Figure 4.9: During this recording, all of the gestures were recorded the same amount of time. The variability in number of video samples in some gestures is due to post-processing filters.



(a) Accuracy results.



(b) Loss results.

Figure 4.10: Accuracy and loss results throughout the pre-training of the dynamic gesture recognition model on the selected 47 gestures of the ChaLearn LSSI dataset.

4.7 Model Training

In this section, the training characteristics and scheme will be detailed. Several experiments were designed and run to assess the performance of the dynamic gesture recognition framework. The goal of these experiments is to test the robustness of the pipeline when trained under different conditions, on different subjects, and when these conditions and subjects are mixed. The details of these experiments are explained on the subsequent paragraphs.

4.7.1 Training Scheme and Specifications

To carry out the training scheme and capitalize on the benefits of transfer learning, we are using the pre-trained model obtained in section 4.6. Starting from this base, each training performed on this section runs for an additional 6,000 steps. A learning rate of 1e-3 is used on a Stochastic Gradient Descent (SGD) optimizer, along a momentum value of 0.9.

The training scheme for these experiments is divided in three phases, each lasting for 2,000 steps. During the first phase, the parameters of the projection layer and the LSTM block are frozen. This allows the model to focus training on the classifier layer. On the second phase, the LSTM block is unfrozen and begins training. Finally, at step 4,000, the projection layer also unfreezes and begins training. Thus, during the last 2,000 steps, the whole model is being trained simultaneously.

4.8 Experimental Results

This section will present the result of several experiments performed with the model. To start with this section, the validation of the model will be presented. Afterwards, subject dependent model experiment results are shown. Then, subject independent experiments are shown with the model that achieved most promising results. Finally, since we observed that the training for some of these experiments may have been too short, we present the results of a trained model for an extended period of time. A summary of the results of these experiments are shown in Tables 4.2 and 4.3.

4.8.1 Model Validation

The first goal for the dynamic gesture recognition framework is to determine whether it is capable of learning features on diver gestures or not. To this end, the model is trained on our recorded diver gesture dataset. From each subject, 40 video samples were selected for the test split to ensure that testing is performed equally on all subjects. For each gesture, 2 samples were selected on each subject. As a result, approximately 20% of our dataset is divided into the test split, while the remaining videos were labeled for training.

Model validation was carried out by training the model with data from all subjects (A, B, C, and D). The training for this experiment follows the general specifications listed in 4.7.1. During training, the testing of this experiment is classified by subset. It was tested on both subjects of the underwater subset, labeled as “Underwater Subset Test Split” on Figures 4.11 and 4.12, and on both subjects of the air subset, labeled as “Air Subset Test Split”. Additionally, two Confusion Matrices are presented in Figure 4.13, following the same subset classification.

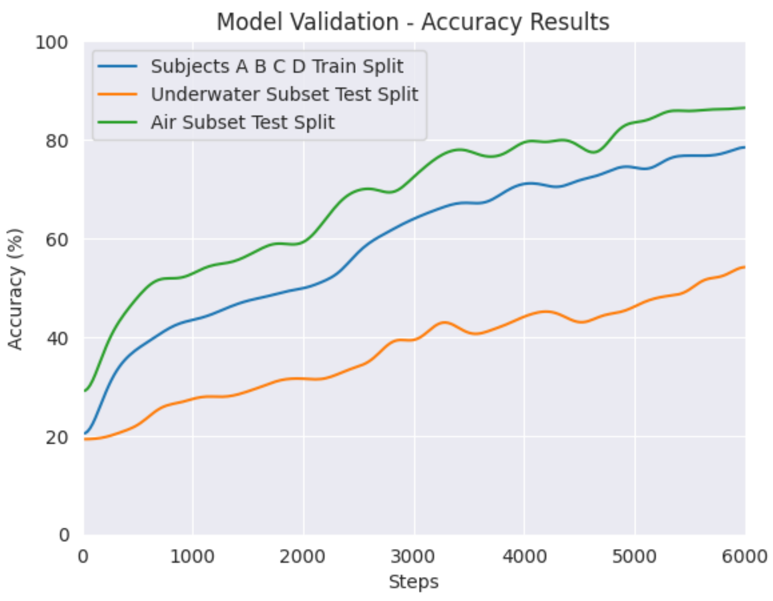


Figure 4.11: Accuracy results throughout the training for the model validation. Blue: accuracy achieved on the train data used. Orange: accuracy achieved on the test split for the underwater subset (both subjects). Green: accuracy achieved on the test split for the air subset (both subjects).

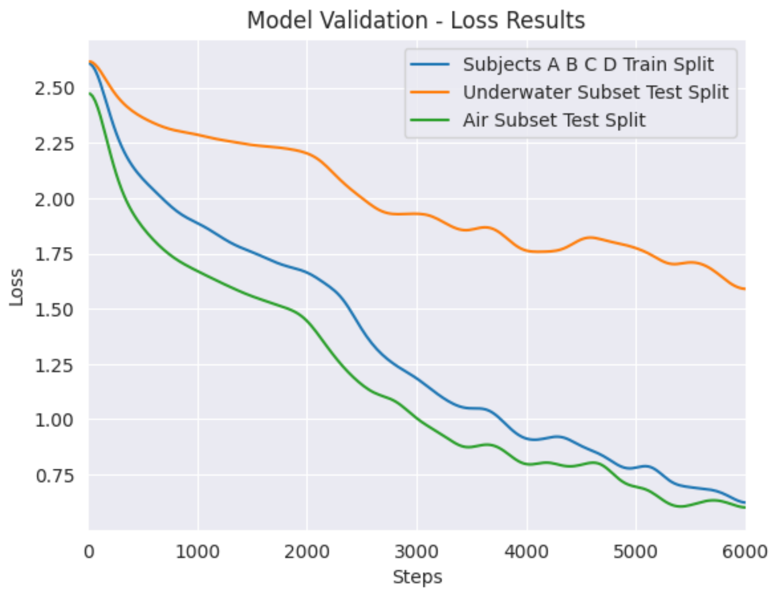
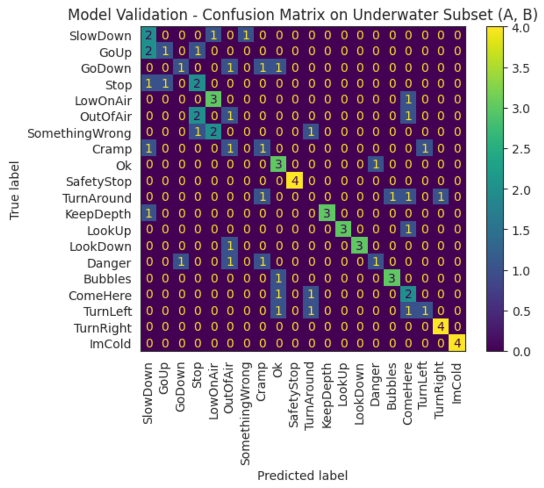


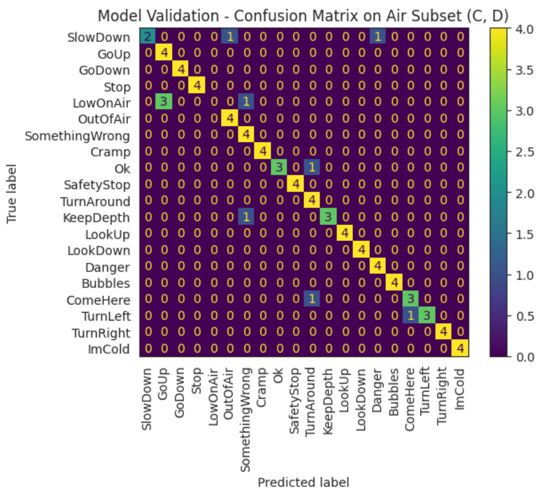
Figure 4.12: Loss results through the training for the model validation. Blue: loss on the train data used. Orange: loss for the test split of the underwater subset (both subjects). Green: loss for the test split of the air subset (both subjects).

As it can be observed in both the Accuracy and Loss Results, the model is capable of improving its identification of dynamic diver gestures of our recorded dataset. It achieves approximately 87.5% accuracy on the “Air Subset Test Split” and just below 60% for the “Underwater Subset Test Split”. Despite being trained on both subsets in parallel, there is a gap in performance when comparing the performance in both environments. A similar behaviour can be observed in the Loss Results, where the curve for the “Air Subset Test Split” is close to the curve of the train split. However, the “Underwater Subset Test Split” loss curve lags behind on the performance.

The confusion matrix on the Air Subset shows how most of the predictions made are correct. How-



(a) Underwater Subset.



(b) Air Subset.

Figure 4.13: Confusion matrices for the model validation. These confusion matrices were plotted against the test splits for each of the depicted subjects. The best trained model for this experiment was used.

ever, it is interesting to note how most of the “Low on Air” gestures as being mislabeled as “Go Up”, which is actually a very similar motion. Other than that, the other mislabeled gestures don’t seem to follow a trend. The Underwater Subset confusion matrix is much more chaotic. This can be associated with the accuracy levels attained at the end of the training. It is important to mention that the Accuracy and Loss graphs show a trend that could continue to improve if given more time. However, to have enough time to train and run all experiments, a 6000 step limit was imposed. Having mentioned this, it is fair to say that the model could still show a better organization in the confusion matrix.

Nonetheless, the first goal that we wanted to achieve with our model is attained. It is demonstrated that the model is capable of learning from our recorded dataset.

4.8.2 Subject Dependent Experiments

The purpose of this subsection is to explore the capabilites of the model when it is tested on the same subject that it is trained. These experimentations are important as it sets a benchmark on what the model can achieve if the end-user dedicates time to recording training videos. This subsection will be divided in two experiments.

First, we will train the models on the air subset in three batches and observe the performance of the model. These batches will be trained as follows:

- Train on subject C;
- Train on subject D;
- Train on subjects C and D.

Afterwards, three models will be trained on three batches of the underwater subset in a similar fashion. The details of the three batches are as follows:

- Train on subject A;
- Train on subject B;
- Train on subjects A and B.

All of these training batches will be performed following the training specifications detailed in 4.7.1.

To provide a contrast, on those models trained on one subject, we will also present the results of the model on the subject that it is not trained. Furthermore, the models trained on the subjects of the air subset will also be tested against the underwater subset. This is interesting due to the fact that it is much easier to record a bigger quantity of videos outside of the water, than inside it.

The accuracy results of the first three batches of experiments, focused on the Air Subset of our dataset, are shown in Figure 4.14. Based on these results, testing on the same train subject can provide good accuracy. For example, the model achieves 80% accuracy when training and testing on subject C. As for subject D, the model achieved 70% accuracy when training and testing on himself. Furthermore, the results are improved when using both subjects of the air subset for training, lifting accuracy up to 85% on the air subset test splits. Nonetheless, even if this is not the focus of these experiments, there is an indication that the results do not translate well to the underwater domain. The accuracy on the underwater subset test splits does not go over 20% in any of the three scenarios.

Even though air-to-water does not seem to transfer that well in the model, air-to-air does achieve promising results, which means the model is capable of generalization in this domain. Moreover, the model is capable of achieving good results on the training subject in this environment.

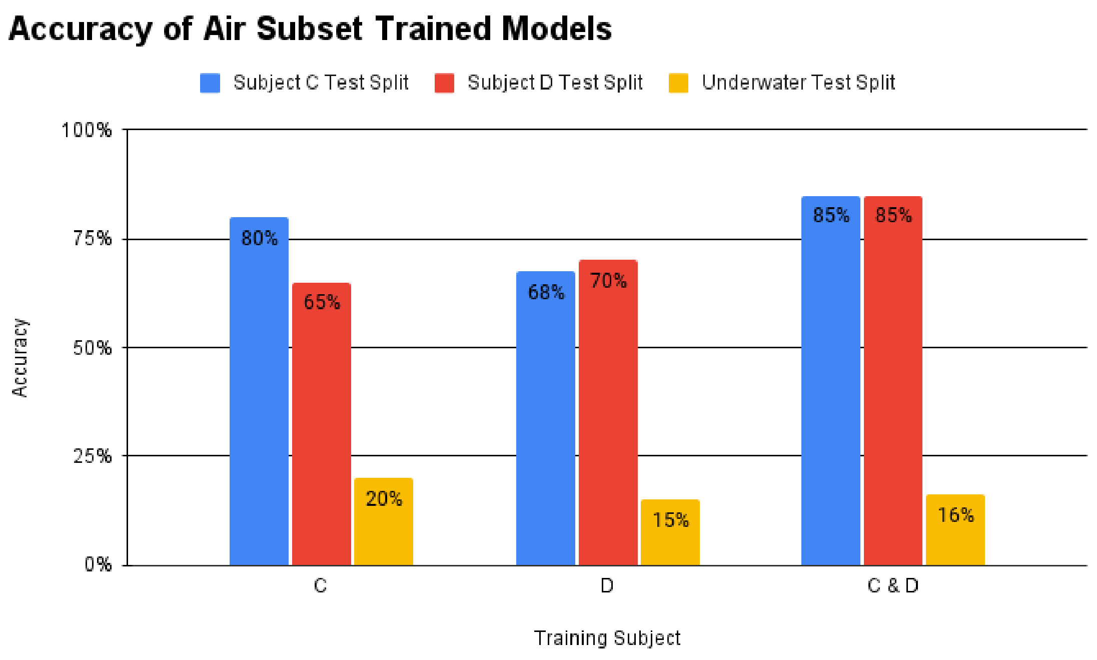


Figure 4.14: Accuracy results of the models trained on the subjects of the Air Subset. The “Underwater Test Split” contains subjects A and B.

Regarding the second group of batches focused on the Underwater Subset of the recorded dataset, the accuracy results are displayed in Figure 4.15. The results are not as good as those observed on the Air Subset. When trained on subject A, the model is only able to achieve a 43% accuracy on itself. However, a better scenario is observed when training on subject B, where the model is able to achieve an accuracy of 68% on itself. Like in the previous batch of experiments, it is not the current focus, but we do observe a gap between the subject that the model is trained on and the unknown subject.

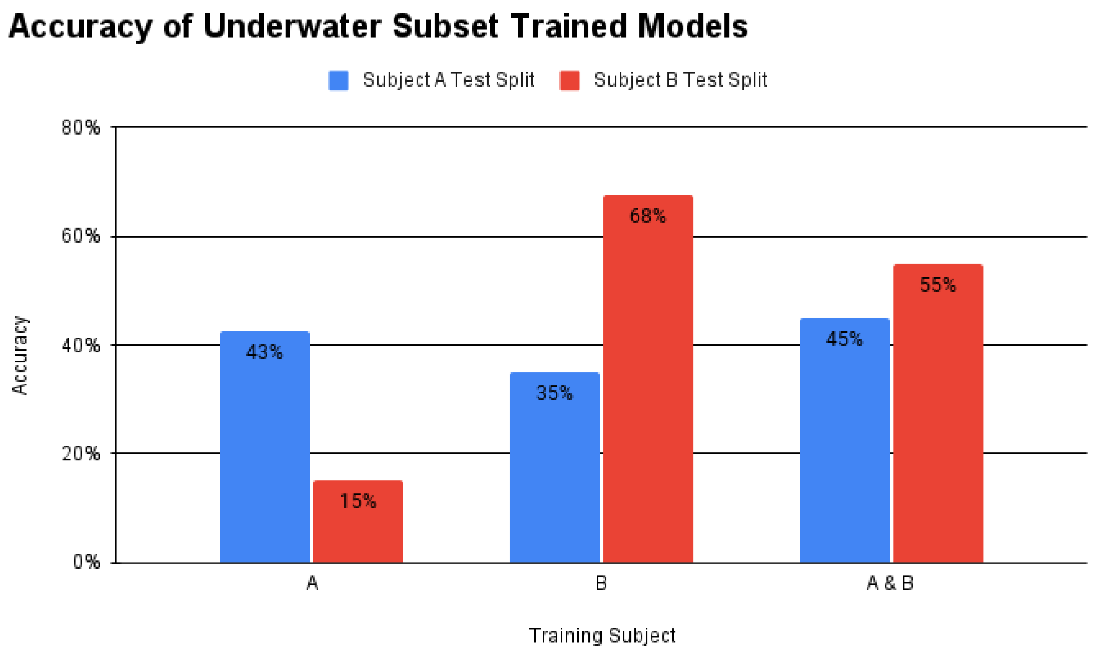


Figure 4.15: Accuracy results of the models trained on the Underwater Subset subjects.

After performing both batches, we can observe a gap between the subject dependent results on the Air Subset and the Underwater Subset. The Air Subset averages an accuracy of 75% when training and testing solely on one subject, while the Underwater Subset averages 55.5%. Three main factors could be influencing this difference:

- Subset Size Difference. The underwater subset is slightly smaller than the air subset, since some of the underwater videos were filtered out during pre-processing and some gestures were not repeated the same number of times.
- Underwater Effect on Off-the-shelf Neural Networks. The human pose estimation model and the hand pose estimation model are not trained with underwater samples. The change in environment could affect negatively their performance, propagating this error throughout the pipeline.
- Gesture Motions. Divers are actively fighting against underwater forces to remain still, which makes it harder for them to repeat pristine motions as in the air medium.

Nonetheless, the air subset results show us that subject dependent models can achieve upwards of 70% accuracy on diver gestures. And big enough dataset of user recorded gestures may provide a better performance on inference.

4.8.3 Subject Independent Experiments

The third group of experiments is set on verifying the ability of our model the perform on subjects it has not observed during training. To this end, four models were trained on the train split of three different subjects. The training was performed following the specifications listed in 4.7.1. Afterwards, each one of these models was tested on the test split of the remaining unused subject. The accuracy achieved by this model is compared to the accuracy achieved by the baseline model, which has been trained on all four subjects. The results of this experiment can be found on Figure 4.16

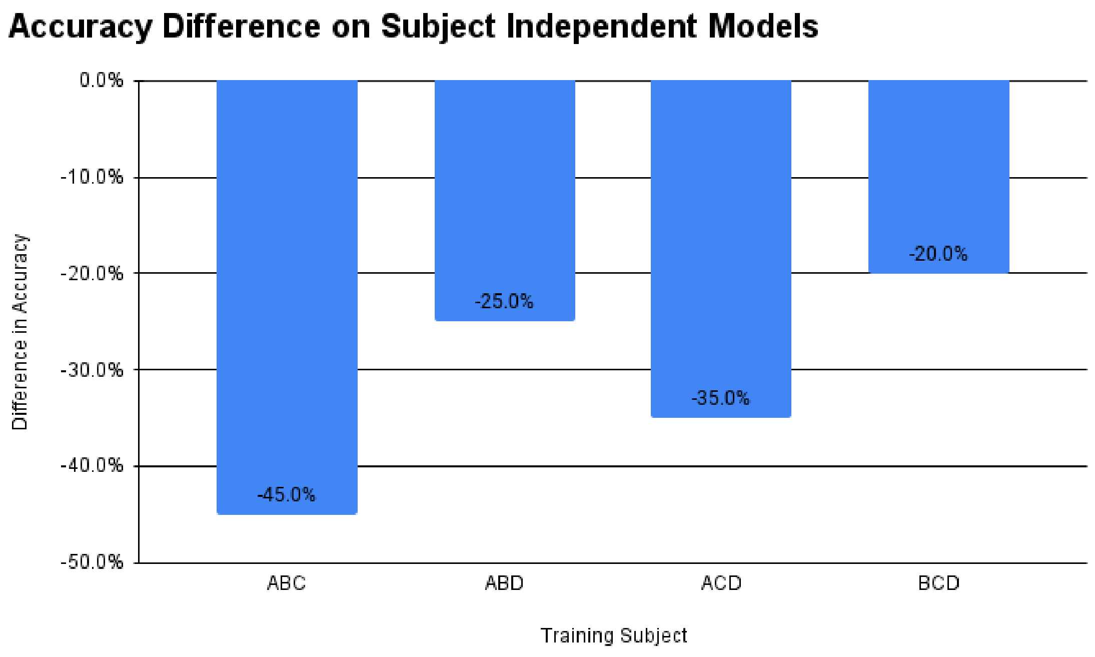


Figure 4.16: Accuracy difference achieved on the independent subject when compared against the model trained on all four subjects.

As it can be observed, there is quite a big performance drop on the accuracy of the model when testing on a new subject. The accuracy drop goes from 25% in the best case scenario to 45% in the worst. It does not seem, based on the current model and dataset size, that the framework works great on unseen subjects.

4.8.4 Extended Model Validation

Since the accuracy and loss curves of the baseline model trained on all four subjects still showed opportunity of improvement, as it can be seen on the previously presented Figures 4.11 and 4.12, the same experiment was repeated for a longer training period. This time, instead of limiting the model to 6,000 steps, the duration of the training was raised to 18,000 steps. The accuracy and loss curves of the training of this model can be observed on Figures 4.17 and 4.18. The confusion matrices of this model on each of the subsets is also presented on Figure 4.19.

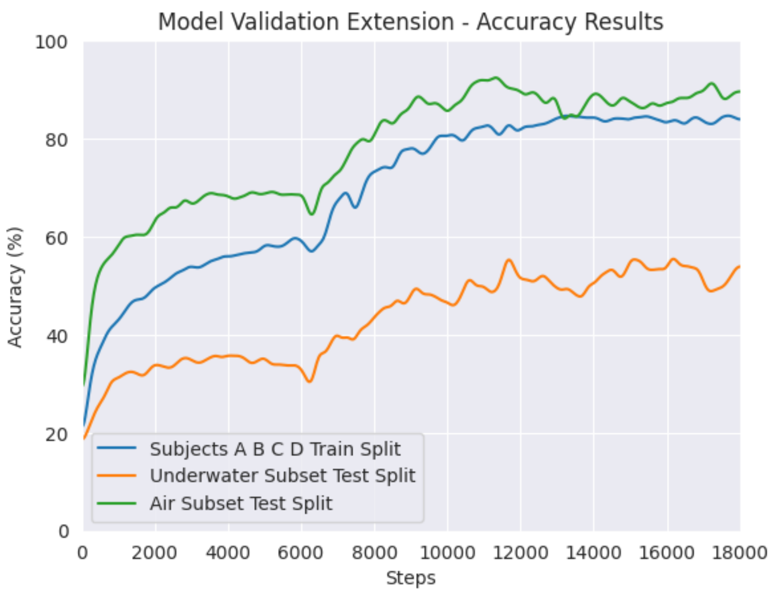


Figure 4.17: Accuracy results throughout the training for the extended model validation. Blue: accuracy achieved on the train data used. Orange: accuracy achieved on the test split for the underwater subset (both subjects). Green: accuracy achieved on the test split for the air subset (both subjects).

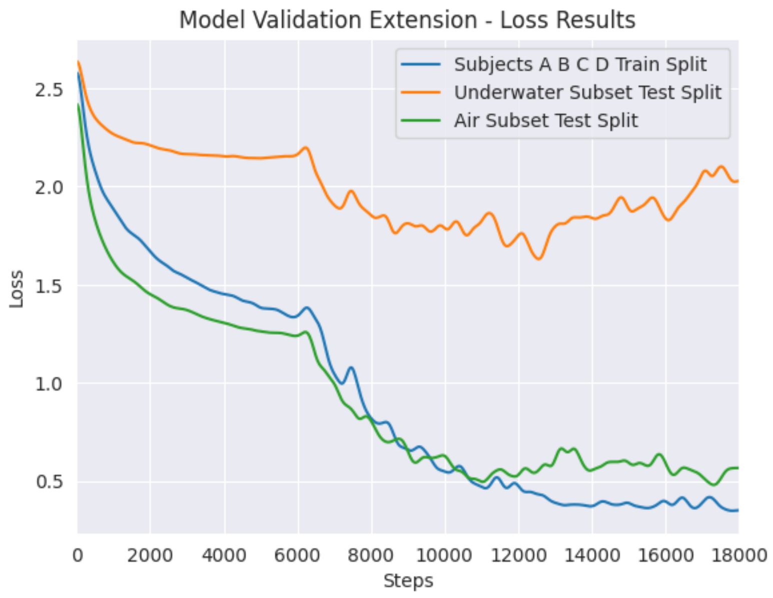
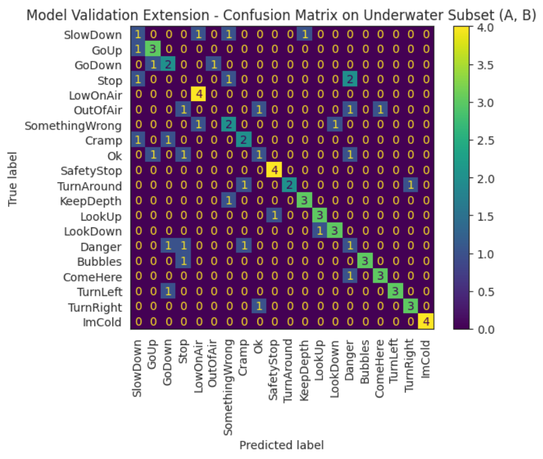


Figure 4.18: Loss results through the training for the extended model validation. Blue: loss on the train data used. Orange: loss for the test split of the underwater subset (both subjects). Green: loss for the test split of the air subset (both subjects).

On this experiment, the curves seem to have reached a plateau by the end of the training. The model achieved a training accuracy of around 84.1%, which is 3 percentile higher than the training on 6,000 steps. Similarly, the accuracy on both subsets have improved. The accuracy on the air subset increased from 87.5% to 90.0%. On the other hand, the accuracy on the underwater subset moved from 52.5% to 58.7%.

Regarding the confusion matrices, despite the improvement achieved by this model, still present a clear difference between both subsets. On the air subset, the “Low on Air” predictions that were mostly incorrect on the 6,000 steps model seem to have improved. On the other hand, the underwater subset


(a) Underwater Subset.

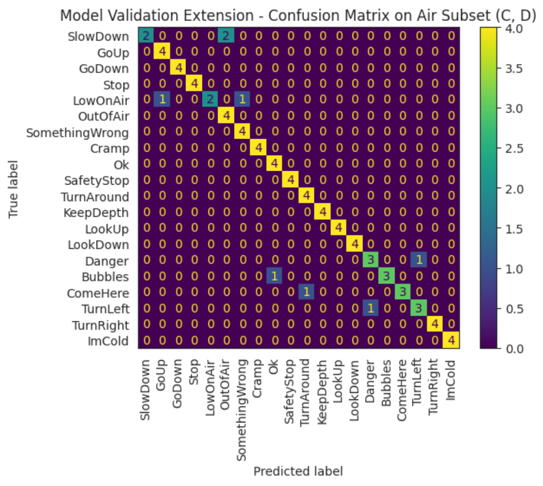

(b) Air Subset.

Figure 4.19: Confusion matrices for the extended model validation. These confusion matrices were plotted against the test splits for each of the depicted subjects. The best trained model for this experiment was used.

shows a slight improvement across the board, but nothing too specific.

The improvements achieved on this experiment show that the performance of the model can yet be improved without changing the architecture. The training procedure can still be improved by fine-tuning the optimizer parameters, the length of the training, batch size, or the training scheme.

4.8.5 Experiment Summary

This subsection summarizes the results obtained for the performed experiments. Two tables are shown: Table 4.2 summarizing the accuracy achieved by each model on different test subjects, and Table 4.3 summarizing the values of the loss function on different test subjects.

Accuracy Results Summary for Experiments

	Accuracy							
Trained On	*Train Split*	*A**	*B**	*C**	*D**	*AB**	*CD**	*ABCD**
A	85.3%	42.5%	15.0%	15.0%	12.5%	28.7%	13.7%	31.2%
B	84.1%	35.0%	**67.5%**	20.0%	10.0%	51.2%	15.0%	33.1%
C	83.4%	22.5%	17.5%	80.0%	65.0%	20.0%	72.5%	46.2%
D	78.9%	20.0%	10.0%	67.5%	70.0%	15.0%	68.7%	41.9%
A & B	81.7%	45.0%	55.0%	25.0%	17.5%	50.0%	21.2%	35.6%
C & D	81.5%	17.5%	15.0%	85.0%	85.0%	16.2%	85.0%	50.6%
A, B & C	58.2%	**57.5%**	52.5%	65.0%	42.5%	55.0%	53.7%	54.4%
A, B & D	82.0%	45.0%	52.5%	62.5%	85.0%	48.7%	73.7%	61.2%
A, C & D	81.8%	42.5%	17.5%	85.0%	**92.5%**	30.0%	88.7%	59.4%
B, C & D	78.65%	32.5%	52.5%	77.5%	77.5%	42.5%	77.5%	60.0%
A, B, C, & D	81.1%	52.5%	52.5%	**87.5%**	87.5%	52.5%	87.5%	70.0%
A, B, C, & D **	84.1%	50.0%	**67.5%**	**87.5%**	**92.5%**	**58.7%**	**90.0%**	**74.4%**

Table 4.2: This table summarizes the accuracy results for the experiments performed. An * denotes that the test was performed on the test split of this, or these, subject. ** means that this model was trained for 3 times longer (18,000 steps).

Loss Results Summary for Experiments

	Loss							
Trained On	*Train Split*	*A**	*B**	*C**	*D**	*AB**	*CD**	*ABCD**
A	0.3743	2.5614	4.4651	4.3387	3.9645	3.5132	4.1516	3.8324
B	0.3886	3.7484	1.4353	4.2152	4.8893	2.5919	4.5522	3.5721
C	0.4007	4.2793	4.7367	0.8082	1.4225	4.5080	1.1153	2.8117
D	0.5716	4.4120	5.8872	1.6262	1.1712	5.1495	1.3985	3.2740
A & B	0.5224	1.9335	1.5649	3.2028	3.3420	1.7493	3.2722	2.5101
C & D	0.4805	4.2517	4.7551	0.7099	0.5156	4.5035	0.6128	2.5582
A, B & C	1.2620	1.5652	1.3726	1.2366	1.8744	**1.4689**	1.5555	1.5122
A, B & D	0.7622	**1.3726**	1.6026	0.9562	0.8448	1.8377	0.9006	1.3692
A, C & D	0.4295	2.2502	3.3595	0.5860	**0.4617**	2.8048	**0.5239**	1.6643
B, C & D	0.6496	2.8930	1.6546	0.8532	0.7895	2.2737	0.8213	1.5475
A, B, C, & D	0.9863	1.7557	1.4149	0.6650	0.5996	1.5853	0.6322	**1.1088**
A, B, C, & D **	0.3705	2.2671	**1.1796**	**0.5829**	0.5027	1.7233	0.5428	1.1331

Table 4.3: This table summarizes the loss results for the experiments performed. An * denotes that the test was performed on the test split of this, or these, subject. ** means that this model was trained for 3 times longer (18,000 steps).

4.9 Live Demonstration

Using the aforementioned code implementations, a demo was developed to perform live offline tests of the dynamic gesture recognition model. The layout of the demo can be observed in figure 4.20. The demo is run locally and connects to a virtual environment with a BlueROV simulation in the servers of DSOR lab. The user can perform gestures to a webcam or a video can be supplied on demand, which are processed and classified by our dynamic gestures model. Then, according to the identified gesture, the corresponding command is sent to the simulation.

This allows us to effectively command a BlueROV through underwater diver gestures. A set of examples can be observed in figure 4.21.

4.10 Vehicle Operation Software Stack

In the last section of this work, we will present an implementation of human-robot collaboration with a BlueROV2 Heavy, using our gesture recognition model, that works with both the actual vehicle and the simulated vehicle. In the following paragraphs, the scope, the framework, the implementation, and the results of the simulation will be explained. As mentioned previously, the high-level vehicle operation software stack from DSOR will be used to begin to encode experimental motion primitives into the BlueROV system. FAROL is being used as the main block used for the control of the underwater vehicle. Gazebo is the chosen alternative for 3D simulations with the underwater vessel.

Since the goal of this research work is focused on the possibility of identifying underwater diver gestures through a monocular camera to communicate with a ROV, a simple set of motion primitives will be used for testing. A bigger group of possible primitives was listed on Chapter 2 to paint a broader picture

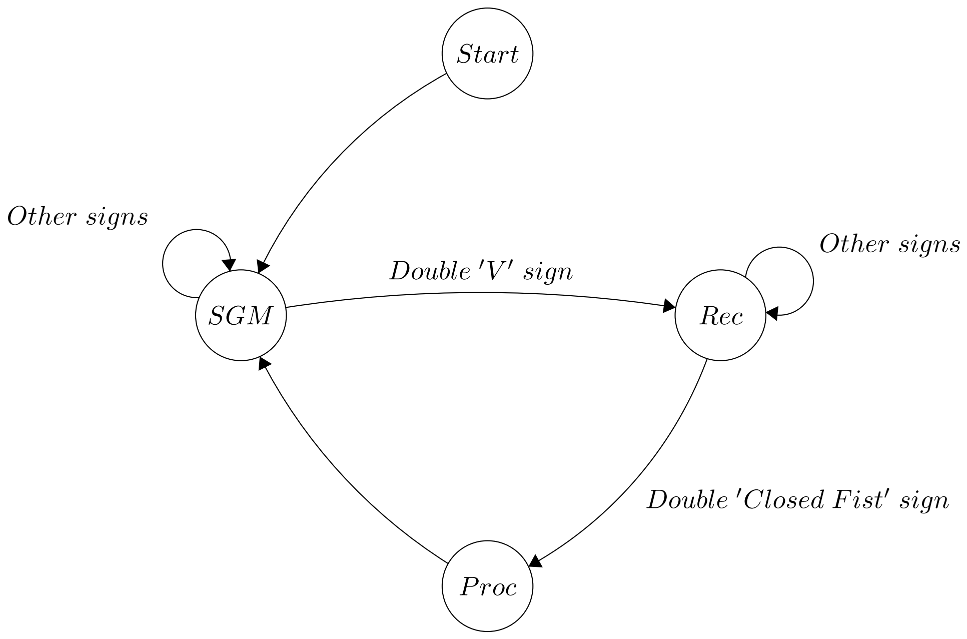


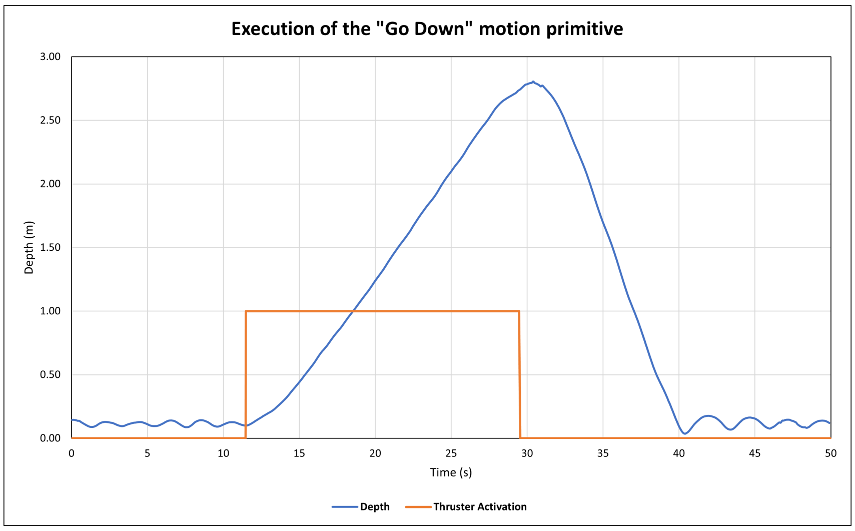
Figure 4.20: Finite State Machine of the live demo. Model initialization is performed in the *Start* state. Once everything is running, the demo moves to the *Static Gesture Mask* state, where it waits for the double 'V' activation signs. Once it receives the correct gestures, it moves to the *Rec* state, where video is recorded until the user performs the double 'Closed Fist' gestures. Once the demo stops recording, it processes the video and performs an inference on it. Lastly, it goes back to the *SGM* state.

and exemplify the opportunities in the field. From Table 2.1, the "Rotate" and "GoDown" motion primitives were selected. Likewise, they were paired with the "Turn Around" and "Go Down" diver gestures, respectively.

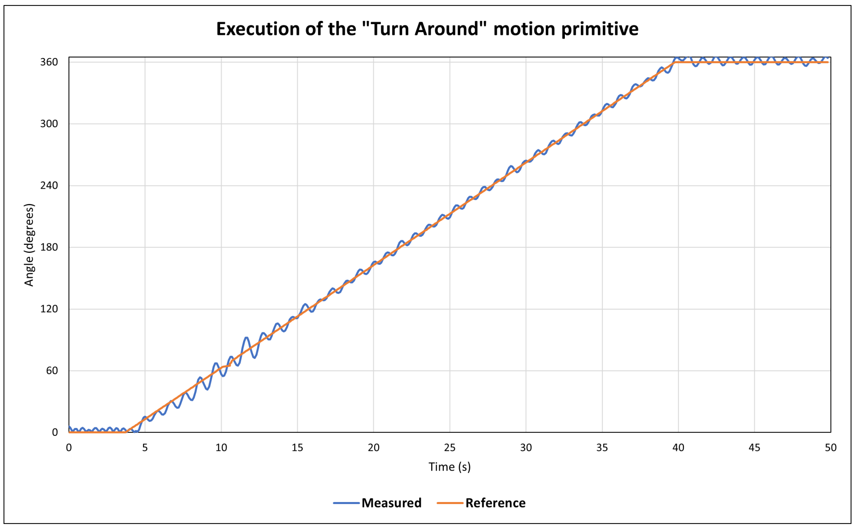
We based the simulations on the framework developed by DSOR in IST to perform this experiment. The provided framework contains the functionality to simulate the environment, simulate the BlueROV 2 vehicle, its sensors and actuators, the control system and algorithm, and a communications interface. This framework was developed on ROS, making it very modular and shareable online with the research community. To integrate our solution to this framework, a script was developed with *roslibpy*, a library that enables communications between python scripts and ROS nodes.

Since we don't have the possibility of performing gestures underwater in real-time, sample videos from the test split of the underwater subset were selected. The python script begins by setting up the ROS interface, loading the gesture recognition model, and initializing communications. Once it is done with this phase, it waits for user input to load the desired video sample for the model to perform inference on. After inference is finished, the model determines the identified gesture. Based on this result, the script publishes a ROS Message with the correspondent instructions. The execution of these two motion samples can be observed in the Figure 4.21.

As observed, once the model identifies the gesture and the script commands the execution of the motion primitive, the BlueROV2 succesfully performs the maneuver before going back to the rest position.



(a) ”GoDown” motion primitive.



(b) ”Rotate” motion primitive.

Figure 4.21: BlueROV2 Heavy executing motion primitives. (a) shows the depth increase, in blue, and the reference signal, in orange, after the gesture is detected. (b) plots the variation in the yaw angle, in blue, and the reference yaw angle, in orange, after the gesture is detected.

Chapter 5

Conclusions

This chapter concludes the developed research. It summarizes observations on the work done and it also proposes possible future work to be done.

5.1 Closing Observations

This research focused on the problem of underwater human-robot communication, proposing a vision-based solution for dynamic gesture identification and classification. As it was explained in Chapter 2, there had been multiple different approaches to tackle this problem, but no clear solution had been found. There is no one-size-fits-all solution, it depends on the characteristics of the environment and the tools at disposal. The work developed here is not an exception, but it provides yet a new tool that may be further researched for possible deployments.

In order to tackle the problem of human-robot communication, a pipeline was developed to exploit the capabilities of modern state-of-the-art neural network models. This pipeline was detailed in Chapter 3 and it is divided into two sections: the Spatial Attention Module and the Neural Network Architecture. The implementation of both parts will be discussed in the following paragraphs, providing some key points that may be improved.

The implementation of the Spatial Attention Module brought to light some issues that may be improved upon to achieve better results with the dynamic gesture recognition model:

1. **Human Detection Model.** Despite using state-of-the-art neural network models for human detection, these off-the-shelf models do not obtain the same results underwater as they do on a normal environment. As such, their performance can be improved by fine-tuning them with an underwater dataset. By improving the performance of this model, it will be possible to reduce misdetections or lack of detections of divers. Since this is one of the first steps performed in the pipeline, its performance is key to achieving good results throughout the process. Namely, this directly improves the result of the pose estimation model.

2. **Pose Estimation Model.** Similarly to the Human Detection Model case, it has proven difficult for off-the-shelf models to correctly estimate pose keypoints from a diver in in some underwater scenarios, specially as light conditions get harsher and the diver is in full black suit. A proper training phase to adequate the model for these scenarios might reduce noise and improve confidence on the inferences. The estimation of these keypoints directly impact the pose vector that the dynamic gesture recognition model receives as input.

3. **Hand Pose Estimation Model.** To finish addressing the models, a similar issue arises for the Hand Pose Estimation model. It provides quite a good estimation when the diver is bare-handed, but the results vary when the diver is wearing gloves. This model impacts the hand extraction algorithm, which might provide an incomplete hand if the estimation is off.

4. **Hand Tracking Algorithm.** Lastly, the hand tracking algorithm can also be improved. While it was not a frequent issue, there were some scenarios where the left hand was identified as the right hand, and vice-versa. This becomes a problem because it breaks the continuity of the gesture being performed.

Regarding the Neural Network Architecture, there are also some points that could have be improved upn, but could not be done in the scope of this research.

1. **Convolutional Neural Network.** The used CNN model could be improved, as the InceptionV3 model is not considered state of the art anymore. Improving this could bring about faster processing times and an extraction of an embedding with more meaningful features.

2. **LSTM Block.** The implementation of the LSTM block to learn features across time steps proved succesful, nonetheless it was also proposed to change the architecture of the model to implement a Transformer-based approach. This would substitute the LSTM block to process the augmented vectors for the length of the execution of a gesture.

Nonetheless, despite these obstacles, the results shown in Chapter 4 are quite promising, as the experiments achieved a good performance, even while training the model on a limited dataset.

5.2 Future Work

To finish this chapter, this section will briefly discuss possible future work that directly or indirectly relates to this subject. A summary of proposed points is detailed here:

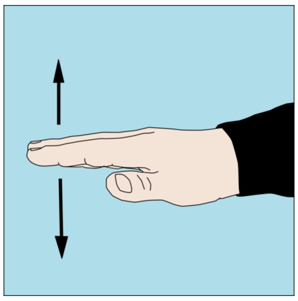
1. **Construct meaningful, open, and multidisciplinary underwater datasets.** One of the main challenges we faced when working on this research was the lack of adequate underwater data for artificial models. It would be tremendously helpful for future endeavors by the community if relevant datasets are built and shared for the most common tasks in AI.

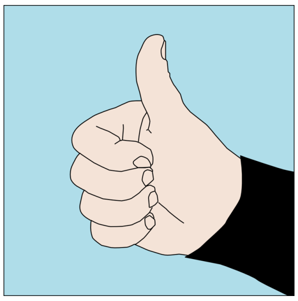
2. **Implement augmented hand pose vectors instead of hand embeddings.** Since a model is already implemented to extract the hand keypoints, it would be interesting to observe the performance of the framework if an augmented hand pose vector is used.

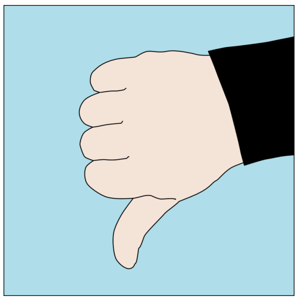
3. **Implement a Transformer.** As previously mentioned, it would be interesting to see the performance of a transformer to process the augmented pose vectors of the complete gesture.

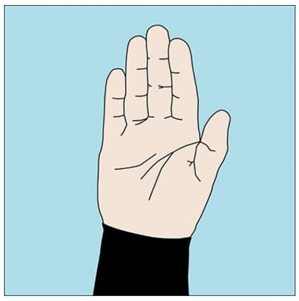
Annex A

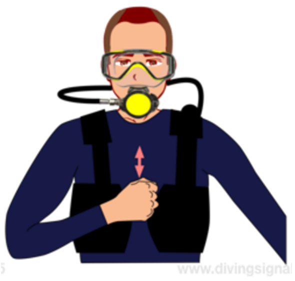
Underwater Gesture Dataset Classes

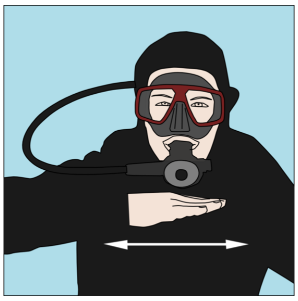

(a) Slow down.

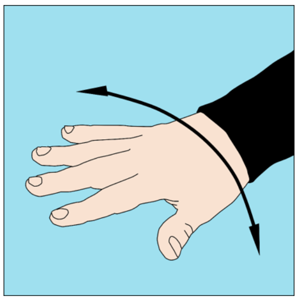

(b) Go up.

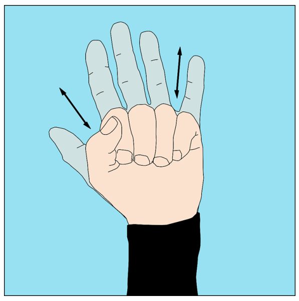

(c) Go down.

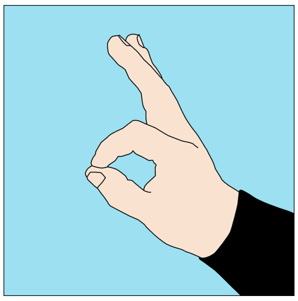

(d) Stop.

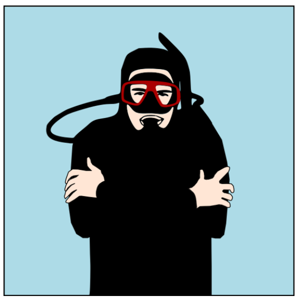

(e) Low on air.

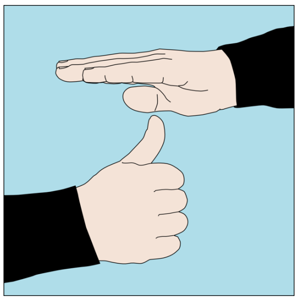

(f) Out of air.

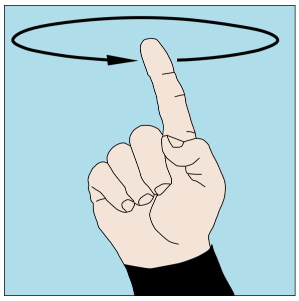

(g) Something wrong.

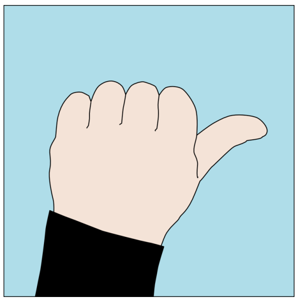

(h) Cramp.

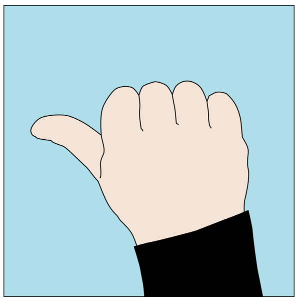

(i) Ok.

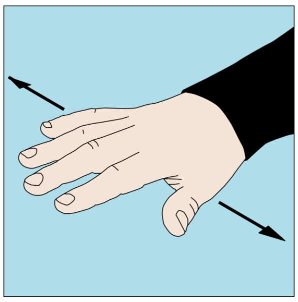

(j) I'm cold.

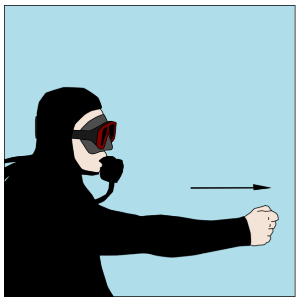

(k) Safety stop.

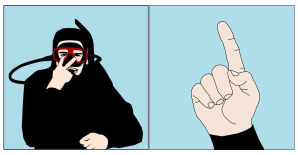

(l) Turn around.

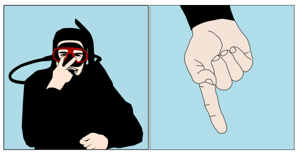

(m) Go right.

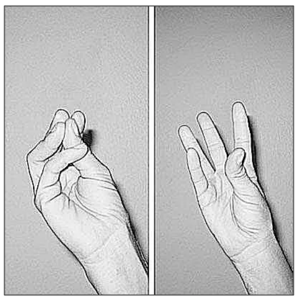

(n) Go left.


(o) Keep your depth.


(p) Danger.


(q) Look up.


(r) Look down.


(s) Bubbles/leak.

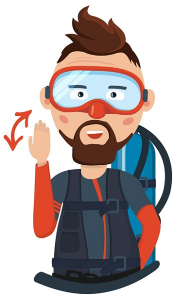

(t) Come here.

Figure A.1: List of 20 gestures selected for our underwater gestures dataset.

Bibliography

[1] K. Sun, W. Cui, and C. Chen, “Review of underwater sensing technologies and applications,” *Sensors*, vol. 21, p. 7849, 11 2021.

[2] M. Lin and C. Yang, “Ocean observation technologies: A review,” *Chinese Journal of Mechanical Engineering*, vol. 33, p. 32, 12 2020.

[3] C. Whitt, J. Pearlman, B. Polagye, F. Caimi, F. Muller-Karger, A. Copping, H. Spence, S. Madhusudhana, W. Kirkwood, L. Grosjean, B. M. Fiaz, S. Singh, S. Singh, D. Manalang, A. S. Gupta, A. Maguer, J. J. H. Buck, A. Marouchos, M. A. Atmanand, R. Venkatesan, V. Narayanaswamy, P. Testor, E. Douglas, S. de Halleux, and S. J. Khalsa, “Future vision for autonomous ocean observations,” *Frontiers in Marine Science*, vol. 7, 9 2020.

[4] F. Bonin-Font and A. Burguera, “Towards multi-robot visual graph-slam for autonomous marine vehicles,” *Journal of Marine Science and Engineering*, vol. 8, p. 437, 6 2020.

[5] Y. Qiao, J. Yin, W. Wang, F. Duarte, J. Yang, and C. Ratti, “Survey of deep learning for autonomous surface vehicles in the marine environment,” 10 2022.

[6] Z. Li, Z. Peng, Z. Zhang, Y. Chu, C. Xu, S. Yao, Ángel F. García-Fernández, X. Zhu, Y. Yue, A. Levers, J. Zhang, and J. Ma, “Exploring modern bathymetry: A comprehensive review of data acquisition devices, model accuracy, and interpolation techniques for enhanced underwater mapping,” *Frontiers in Marine Science*, vol. 10, 5 2023.

[7] A. Ito, H. Shiobara, M. Miller, H. Sugioka, J. Ojeda, C. Tassara, M. Shinohara, M. Kinoshita, and H. Iwamori, “Long-term array observation by ocean bottom seismometers at the chile triple junction,” *Journal of South American Earth Sciences*, vol. 124, p. 104285, 4 2023.

[8] B. Li, B. Moridian, and N. Mahmoudian, “Autonomous oil spill detection: Mission planning for asvs and auvs with static recharging,” in *OCEANS 2018 MTS/IEEE Charleston*, 2018, pp. 1–5.

[9] L. Poggi, T. Gaggero, M. Gaiotti, E. Ravina, and C. M. Rizzo, “Recent developments in remote inspections of ship structures,” *International Journal of Naval Architecture and Ocean Engineering*, vol. 12, pp. 881–891, 2020.

[10] M. J. Islam, “Machine vision for improved human-robot cooperation in adverse underwater conditions,” 10 2019.

[11] N. Mišković, A. Pascoal, M. Bibuli, M. Caccia, J. A. Neasham, A. Birk, M. Egi, K. Grammer, A. Marroni, A. Vasilijević, and Z. Vukić, “Caddy project, year 2: The first validation trials**this work is supported by the european commission under the fp7-ict project ”caddy - cognitive autonomous diving buddy” grant agreement no. 611373.” *IFAC-PapersOnLine*, vol. 49, pp. 420–425, 2016.

[12] M. J. Islam, M. Ho, and J. Sattar, "Dynamic reconfiguration of mission parameters in underwater human-robot collaboration," 9 2017.

[13] B. A. Erol, C. Wallace, P. Benavidez, and M. Jamshidi, "Voice activation and control to improve human robot interactions with iot perspectives." IEEE, 6 2018, pp. 1–5.

[14] M. Shridhar and D. Hsu, "Interactive visual grounding of referring expressions for human-robot interaction," 6 2018.

[15] S. M. Chacko and V. Kapila, "An augmented reality interface for human-robot interaction in unconstrained environments." IEEE, 11 2019, pp. 3222–3228.

[16] O. Mazhar, S. Ramdani, B. Navarro, R. Passama, and A. Cherubini, "Towards real-time physical human-robot interaction using skeleton information and hand gestures." IEEE, 10 2018, pp. 1–6.

[17] A. Birk, "A survey of underwater human-robot interaction (u-hri)," *Current Robotics Reports*, 9 2022.

[18] N. Miskovic, A. Pascoal, M. Bibuli, M. Caccia, J. A. Neasham, A. Birk, M. Egi, K. Grammer, A. Marroni, A. Vasilijevic, D. Nad, and Z. Vukic, "Caddy project, year 3: The final validation trials." IEEE, 6 2017, pp. 1–5.

[19] D. Chiarella, M. Bibuli, G. Bruzzone, M. Caccia, A. Ranieri, E. Zereik, L. Marconi, and P. Cutugno, "A novel gesture-based language for underwater human–robot interaction," *Journal of Marine Science and Engineering*, vol. 6, p. 91, 8 2018.

[20] M. J. Islam, M. Ho, and J. Sattar, "Understanding human motion and gestures for underwater human-robot collaboration," *Journal of Field Robotics*, vol. 36, pp. 851–873, 8 2019.

[21] F. Gustin, I. Rendulic, N. Miskovic, and Z. Vukic, "Hand gesture recognition from multibeam sonar imagery**this work has been done within the scope of caddy, a collaborative project funded by the european community's seventh framework programme fp7-challenge 2: Cognitive systems and robotics-under grant agreement 611373." *IFAC-PapersOnLine*, vol. 49, pp. 470–475, 2016.

[22] C. Zheng, W. Wu, C. Chen, T. Yang, S. Zhu, J. Shen, N. Kehtarnavaz, and M. Shah, "Deep learning-based human pose estimation: A survey," 12 2022.

[23] Michael, B. Serge, H. James, P. Pietro, R. Deva, D. Piotr, Z. C. L. L. Tsung-Yi, and Maire, "Microsoft coco: Common objects in context," Tomas, S. Bernt, T. T. F. David, and Pajdla, Eds. Springer International Publishing, 2014, pp. 740–755, *Dataset:* https://cocodataset.org/#home.

[24] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele.

[25] J. Wu, H. Zheng, B. Zhao, Y. Li, B. Yan, R. Liang, W. Wang, S. Zhou, G. Lin, Y. Fu, Y. Wang, and Y. Wang, "Large-scale datasets for going deeper in image understanding." IEEE, 7 2019, pp. 1480–1485.

[26] J. Li, C. Wang, H. Zhu, Y. Mao, H.-S. Fang, and C. Lu, "Crowdpose: Efficient crowded scenes pose estimation and a new benchmark," 12 2018.

[27] Y. Wang, C. Peng, and Y. Liu, "Mask-pose cascaded cnn for 2d hand pose estimation from single color image," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, pp. 3258–3268, 11 2019.

[28] G. Moon, S. i Yu, H. Wen, T. Shiratori, and K. M. Lee, "Interhand2.6m: A dataset and baseline for 3d interacting hand pose estimation from a single rgb image," 8 2020.

[29] S. Jin, L. Xu, J. Xu, C. Wang, W. Liu, C. Qian, W. Ouyang, and P. Luo, “Whole-body human pose estimation in the wild,” 7 2020.

[30] T. D. Nguyen and M. Kresovic, “A survey of top-down approaches for human pose estimation,” 2 2022.

[31] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask r-cnn.” IEEE, 10 2017, pp. 2980–2988.

[32] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun, “Cascaded pyramid network for multi-person pose estimation,” 11 2017.

[33] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, “Openpose: Realtime multi-person 2d pose estimation using part affinity fields,” 12 2018.

[34] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, M. Elgharib, P. Fua, H.-P. Seidel, H. Rhodin, G. Pons-Moll, and C. Theobalt, “Xnect: Real-time multi-person 3d motion capture with a single rgb camera,” 7 2019.

[35] M. Contributors, “Openmmlab pose estimation toolbox and benchmark,” https://github.com/open-mmlab/mmpose, 2020.

[36] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu, Z. Zhang, D. Cheng, C. Zhu, T. Cheng, Q. Zhao, B. Li, X. Lu, R. Zhu, Y. Wu, J. Dai, J. Wang, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, “MMDetection: Open mmlab detection toolbox and benchmark,” *arXiv preprint arXiv:1906.07155*, 2019.

[37] Y. Xu, J. Zhang, Q. Zhang, and D. Tao, “Vitpose: Simple vision transformer baselines for human pose estimation,” 4 2022.

[38] T. Jiang, P. Lu, L. Zhang, N. Ma, R. Han, C. Lyu, Y. Li, and K. Chen, “Rtmpose: Real-time multi-person pose estimation based on mmpose,” 3 2023.

[39] C. Lyu, W. Zhang, H. Huang, Y. Zhou, Y. Wang, Y. Liu, S. Zhang, and K. Chen, “Rtmdet: An empirical study of designing real-time object detectors,” 12 2022.

[40] Y. Li, S. Yang, P. Liu, S. Zhang, Y. Wang, Z. Wang, W. Yang, and S.-T. Xia, “Simcc: a simple coordinate classification perspective for human pose estimation,” 7 2021.

[41] H. Yu, Y. Xu, J. Zhang, W. Zhao, Z. Guan, and D. Tao, “Ap-10k: A benchmark for animal pose estimation in the wild,” 8 2021.

[42] N. Santavas, I. Kansizoglou, L. Bampis, E. Karakasis, and A. Gasteratos, “Attention! a lightweight 2d hand pose estimation approach,” 1 2020.

[43] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, “Real-time continuous pose recovery of human hands using convolutional networks,” *ACM Transactions on Graphics*, vol. 33, August 2014.

[44] D. Tang, T.-H. Yu, and T.-K. Kim, “Real-time articulated hand pose estimation using semi-supervised transductive regression forests.” IEEE, 12 2013, pp. 3224–3231.

[45] S. Yuan, Q. Ye, G. Garcia-Hernando, and T.-K. Kim, “The 2017 hands in the million challenge on 3d hand pose estimation,” 7 2017.

[46] A. Armagan, G. Garcia-Hernando, S. Baek, S. Hampali, M. Rad, Z. Zhang, S. Xie, M. Chen, B. Zhang, F. Xiong, Y. Xiao, Z. Cao, J. Yuan, P. Ren, W. Huang, H. Sun, M. Hrúz, J. Kanis, Z. Krňoul, Q. Wan, S. Li, L. Yang, D. Lee, A. Yao, W. Zhou, S. Mei, Y. Liu, A. Spurr, U. Iqbal, P. Molchanov, P. Weinzaepfel, R. Brégier, G. Rogez, V. Lepetit, and T.-K. Kim, "Measuring generalisation to unseen viewpoints, articulations, shapes and objects for 3d hand pose estimation under hand-object interaction," 3 2020.

[47] J. Cheng, Y. Wan, D. Zuo, C. Ma, J. Gu, P. Tan, H. Wang, X. Deng, and Y. Zhang, "Efficient virtual view selection for 3d hand pose estimation," 3 2022.

[48] M. Rezaei, R. Rastgoo, and V. Athitsos, "Trihorn-net: A model for accurate depth-based 3d hand pose estimation," 6 2022.

[49] O. Mazhar, S. Ramdani, and A. Cherubini, "A deep learning framework for recognizing both static and dynamic gestures," 6 2020.

[50] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[51] Y. Chen, L. Zhao, X. Peng, J. Yuan, and D. N. Metaxas, "Construct dynamic graphs for hand gesture recognition via spatial-temporal attention," in *BMVC*, 2019.

[52] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification." ACM Press, 2006, pp. 369–376.

[53] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 6 2014.

[54] R. Cahuantzi, X. Chen, and S. Güttel, "A comparison of lstm and gru networks for learning symbolic sequences," 7 2021.

[55] O. M. Sincan, J. C. S. J. Junior, S. Escalera, and H. Y. Keles, "Chalearn lap large scale signer independent isolated sign language recognition challenge: Design, results and future research," 5 2021.

[56] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 10 2020.

[57] N. Neverova, C. Wolf, G. W. Taylor, and F. Nebout, "Multi-scale deep learning for gesture detection and localization," pp. 474–490, 2015.

[58] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, "Ntu rgb+d: A large scale dataset for 3d human activity analysis," 4 2016, *Dataset:* https://rose1.ntu.edu.sg/dataset/actionRecognition/.

[59] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. C. Kot, "Ntu rgb+d 120: A large-scale benchmark for 3d human activity understanding," 5 2019, *Dataset:* https://rose1.ntu.edu.sg/dataset/actionRecognition/.

[60] J. Wan, S. Z. Li, Y. Zhao, S. Zhou, I. Guyon, and S. Escalera, "Chalearn looking at people rgb-d isolated and continuous datasets for gesture recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2016, pp. 761–769, *Dataset:* https://chalearnlap.cvc.uab.cat/dataset/21/description/#.

[61] O. Mazhar, “Opensign - kinect v2 hand gesture data - american sign language,” 2019, *Dataset:* https://data.mendeley.com/datasets/k793ybxx7t/1.

[62] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” 12 2015.

[63] A. Mavi and Z. Dikle, “A new 27 class sign language dataset collected from 173 individuals,” 3 2022.

[64] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” 3 2015.

[65] H. J. Escalante, V. Ponce-Lopez, J. Wan, M. A. Riegler, B. Chen, A. Clapes, S. Escalera, I. Guyon, X. Baro, P. Halvorsen, H. Muller, and M. Larson, “Chalearn joint contest on multimedia challenges beyond visual analysis: An overview.” IEEE, 12 2016, pp. 67–73.

[66] F. B. M. J. D. C. F. R. Joao Q., David S., “Free autonomous robots for observations and labelling,” https://github.com/dsor-isr/farol, 2023.

[67] N. Hung, F. Rego, J. Quintas, J. Cruz, M. Jacinto, D. Souto, A. Potes, L. Sebastiao, and A. Pascoal, “A review of path following control strategies for autonomous robotic vehicles: Theory, simulations, and experiments,” *Journal of Field Robotics*, vol. n/a, no. n/a. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.22142