Arm and hand gesture recognition through online learning

A Degree's Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Oscar Pluma Duran

In partial fulfilment
of the requirements for the degree in
AUDIOVISUAL SYSTEMS ENGINEERING

Advisor: Josep Ramon Casas

Barcelona, June 2014
Abstract

The aim of this project is to develop a system capable to move a robot by recognizing the gestures performed by a user. The system has been already implemented by the LIRIS laboratory, and the objective has been to improve its performance and to add more functionalities.

For improving the performance, a restructuring of the already existing feature vectors is proposed. That proposal implies taking into account the kinematic information of the actions when we construct the feature vectors.

And as a new functionality, an online learning has been carried out. It allows the client to update the gesture recognizer and improve its performance in real time. That would allow the client to ameliorate immediately the detection of gestures that have been misclassified.
Resum

L'objectiu d'aquest projecte és desenvolupar un sistema capaç de moure un robot mitjançant el reconeixement de gestos realitzats per l'usuari. El sistema ja ha estat implementat pel laboratori LIRIS, i l'objectiu ha estat millorar les seves prestacions i afegir novel·litats.

Per millorar el rendiment, es proposa una reestructuració dels vectors de característiques ja existents. Aquesta proposta implica tenir en compte la informació cinemàtica de les accions a l'hora de construir els vectors de característiques.

I com a nova funcionalitat, s'ha dut a terme l'aprenentatge en línia, que permet al client actualitzar el classificador de moviments i millorar el seu rendiment en temps real. Això permetria a l'usuari millorar immediatament la detecció de gestos que han estat mal classificats.
Resumen

El objetivo de este proyecto es desarrollar un sistema capaz de mover un robot mediante el reconocimiento de gestos realizados por el usuario. El sistema ya ha sido implementado por el laboratorio LIRIS, y el objetivo ha sido mejorar sus prestaciones y añadir nuevas funcionalidades.

Para mejorar el rendimiento, se propone una reestructuración de los vectores de características ya existentes. Esta propuesta implica tener en cuenta la información cinemática de las acciones a la hora de construir los vectores de características.

Y como nueva funcionalidad, se ha llevado a cabo el aprendizaje en línea, que permite al cliente actualizar el clasificador de movimientos y mejorar su rendimiento en tiempo real. Esto permitiría al usuario mejorar inmediatamente la detección de gestos que han sido mal clasificados.
Acknowledgements

I would like to thank the LIRIS department team for giving me the opportunity to work in this project, which has approached me to the world of computer vision and has allowed me to see a scientific point of view of this topic.

A special thanks to my final project supervisors, Josep Ramon Casas and Christian Wolf, that have oriented me on the steps to be done during the project and the decisions that have been necessary to be taken.

Thanks to Natalia, that has provided to me the huge dataset that she recorded with effort in order to help me to advance in my project.

Thanks to Eric, that has helped me to fix programming problems when I was about to get crazy.
## Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>09/06/2014</td>
<td>Document creation</td>
</tr>
<tr>
<td>1</td>
<td>dd/mm/yyyy</td>
<td>Document revision</td>
</tr>
</tbody>
</table>

### DOCUMENT DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oscar Pluma Duran</td>
<td><a href="mailto:oscar.pluma@alu-etsetb.upc.edu">oscar.pluma@alu-etsetb.upc.edu</a></td>
</tr>
<tr>
<td>Josep Ramon Casas</td>
<td><a href="mailto:josep.ramon.casas@upc.edu">josep.ramon.casas@upc.edu</a></td>
</tr>
<tr>
<td>Christian Wolf</td>
<td><a href="mailto:christian.wolf@liris.cnrs.fr">christian.wolf@liris.cnrs.fr</a></td>
</tr>
</tbody>
</table>

**WRITTEN BY:**
Oscar Pluma Duran  
**Date:** 06/09/2014

**REVIEWED AND APPROVED BY:**
Josep Ramon Casas  
Christian Wolf  
**Date:** dd/mm/yyyy

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oscar Pluma Duran</td>
<td>Project author</td>
</tr>
<tr>
<td>Josep Ramon Casas</td>
<td>Project Supervisor</td>
</tr>
<tr>
<td>Christian Wolf</td>
<td></td>
</tr>
</tbody>
</table>
Table of contents

1. Introduction .................................................................................................................... 9
   1.1. LIRIS Laboratory .......................................................................................... 9
   1.2. Project Objectives ...................................................................................... 10
   1.3. Project Requirements and Specifications ..................................................... 11
   1.4. Incidences .................................................................................................... 12
   1.5. Work Plan ..................................................................................................... 13

2. Project context and Existing solution ........................................................................ 14
   2.1. Machine Learning ........................................................................................ 15
   2.2. Feature Extraction ...................................................................................... 16
   2.3. Classifier – Neural network ........................................................................... 19
   2.4. Training & Prediction Method ........................................................................ 20
   2.5. System architecture ...................................................................................... 21

3. Project development ................................................................................................... 22
   3.1. Contribution ................................................................................................ 22
       3.1.1. Online Training ........................................................................ 23
       3.1.2. New training descriptors & New training vectors ................... 24
   3.2. Block Diagram of the system ....................................................................... 28

4. Tests and Results ....................................................................................................... 33
   4.1. Test methods .............................................................................................. 33
   4.2. Test results ................................................................................................. 35

5. Budget ....................................................................................................................... 38

6. Conclusions and future development ........................................................................ 40
List of Figures

Figure 1. Robot controlled with a tablet ................................................................. 9
Figure 2. Gantt Diagram .......................................................................................... 13
Figure 3. Machine Learning operation scheme ....................................................... 15
Figure 4. Skeleton joints detected by Kinect camera .............................................. 16
Figure 5. The 11 joints used for training. The $\alpha$ angles between joints. .......... 17
Figure 6. Use of sequential pose descriptors in order to form the feature vector ... 18
Figure 7. Sartificial Neural Network ..................................................................... 19
Figure 8. VOIR system classifier structure ............................................................ 20
Figure 9. System Architecture .............................................................................. 21
Figure 10. Recording of new vector labelling by pressing the correspondent key ... 23
Figure 11. Txt file structure for each frame, containing the frame data ............... 24
Figure 12. “session.txt” file structure ................................................................... 25
Figure 13. XML file structure ............................................................................... 26
Figure 14. Accuracy for each training vector type ................................................ 27
Figure 15. Block diagram of the system ................................................................. 28
Figure 16. Dataset partition for each iteration in Cross-Validation ....................... 34
List of Tables:

Table 1. General neural network - Train & Test .............................................................. 35
Table 2. Filter neural network - Train & Test ................................................................... 36
Table 3. General neural network - Cross-Validation ....................................................... 36
Table 4. Filter neural network - Cross-Validation ............................................................ 37
Table 5. Material cost ...................................................................................................... 38
Table 6. Staff cost ........................................................................................................... 38
Table 7. Project total cost ............................................................................................... 39
1. Introduction

1.1. LIRIS LABORATORY

The project is carried out at LIRIS, Laboratoire d'InfoRmatique en Image et Systèmes d'Information.

LIRIS is affiliated to CNRS (Centre National de Recherche Scientifique) under the label UMR 5205; and involves 320 researchers from INSA de Lyon, Université Claude Bernard Lyon 1, Ecole Centrale de Lyon, Université Lumière Lyon 2 and CNRS.

The laboratory is organized in six areas of skills of 20-25 permanents. Each of the 12 research teams belongs to one of these areas:
- Computer Vision and Pattern Recognition
- Geometry and modelling
- Data Science
- Services, Distributed Systems, and Security
- Simulation, virtuality, and computational sciences
- Interactions and cognition

During these last months I have been working in the INSA de Lyon laboratory, that belongs to the Computer Vision and Pattern Recognition area.

Figure 1. Robot controlled with a tablet
One of the LIRIS projects is VOIR, a computer vision platform capable to detect and recognize some human properties in real time. These features recognitions are particularly object detection, face detection, and gesture detection.

My part in this project is to work with the gesture detection improving some of its aspects.

1.2. PROJECT OBJECTIVES

The current system provides acceptable results; but its performance is still not perfect and provides some errors. It is necessary to do some work on stability and reliability.

The given error is due to the following facts:
- There is a lack of a larger training database.
- Misclassification due to the fingers gestures variability.

The objective of the project is to reduce these errors and improve the system's performance by reaching some goals, that are the followings:

1. - Add new type of descriptor to the already existing feature vectors in order to obtain more information about the gestures. That will help the classifier to make better predictions.

2. - Online Learning: Instead of recording a larger training database, the implementation of an online learning on the gesture recognition based in skeleton is proposed.
   In an on online learning system, the user runs the classifier program and can see the gestures that are misclassified. In that case, the user can collect new training data of that gestures while a second user annotates its label on the fly. After that, the classifier is updated immediately using the new recorded descriptors such that the performance of the misclassified gestures is improved.

3. - Hand tracking and Detection of finger gestures: If we add a finger gesture recognition system to the gesture recognition based in skeleton, the fingers gestures variability will be taken into account and the error will be reduced.
1.3. PROJECT REQUIREMENTS AND SPECIFICATIONS

Project requirements:

- The final system is a robot that is moved by human gestures.

- The human gestures that can be detected are 7:
  
  - Stop
  - Scroll: left-to-right
  - Scroll: right-to-left
  - Scroll: up-to-down
  - Scroll: down-to-up
  - Phone: pick up
  - Phone: hang up
  - Hi

- The customer can train new input data by making the pertinent movements and indicating the type of each movement he makes.

Project specifications:

- Gesture classification and image processing is computed in a server instead of in the robot.

- Gesture recognition is computed by a classifier that takes as input the output of 4 sub-classifiers:
  
  - Left hand gesture recognizer
  - Right hand gesture recognizer
  - Skeleton gesture recognizer
  - Speech feature recognizer

- In the two hands gesture recognizers, a finger gestures detector is added in order to improve the system performance.

- Training of new input data is done by online learning: the client indicates the label of each input gestures.
1.4. INCIDENCES

- Implementation of Kinect Interaction failed:

Kinect Interaction is a library available in *Windows Toolkit 1.7* that allows to detect the following hand actions:

- “Grip” of hands (closed hand).
- “Grip Release” of hands (opened hand).
- "Press" (length of stretching the arm).

In order to detect these actions, we follow these steps:

- Extraction of the depth and skeleton streams.
- For each depth and skeleton frame produced by the sensor streams, pass the frame’s data to the appropriate method (*ProcessDepth* or *ProcessSkeleton*) of the interaction stream.
- Obtaining the interaction frame using the processed depth and skeleton streams as an input of the method *GetNextFrame*.
- Reading the data from each interaction frame in order to find out what the user is doing with his hands.

You can work with the Kinect camera two ways: with events or with a timeout. If we work with events, our program runs whenever a new frame representation in the camera is detected. By this way, programming is more difficult.

If we work with a timeout, our program waits some time between each program execution; we can equate the timeout to the camera frame rate. By this way, programming is easier.

The whole VOIR program is implemented by using the timeout because it is simpler. But it turns out that Kinect Interaction only performs perfectly when events are used. After trying unsuccessfully different implementations, we have decided not to use it for the moment because we would need to change the whole VOIR program in order to have a good Kinect Interaction performance.

- System adaptation and classifier training:

  The adaptation of the new implemented system into the existing solution and the classifier training using the new training vector structure have taken more time to perform than we thought. That’s why the finger gesture recognizer has not been implemented.
1.5. WORK PLAN

Due to the both incidences explained in the previous section, the Gantt diagram has changed drastically compared to the project proposal and has the following appearance:

![Gantt Diagram](image-url)

Figure 2. Gantt Diagram
2. **Project context and Existing solution:**

The current gesture detection system allows to move a robot by gesture recognition. For this purpose, a Microsoft Kinect camera is used, which provides the necessary information to predict the gesture to a remote server by applying a machine learning method.

Machine learning method consists in training a system that can learn from data. During the training, it learns to distinguish between classes (in our case, between gestures) from characteristics that define each class in a discriminatory way. These characteristics are called feature vectors and its values have to be as different for each class as possible so that the machine learning can distinguish each class by seeing the information stored in the feature vectors.

So, when new feature vectors are used as an input for the machine learning system, the classifier can predict the class that they are describing by comparing the input feature values with the feature values viewed in the training session. When the class is predicted, a label is assigned to the input data.
2.1. Machine learning

For the machine learning training implementation, we acquire certain feature vectors from a Kinect camera (explained in 2.2. Feature Extraction section). After that, all descriptors are labelled manually by an expert.

At present, a large dataset has been acquired and labelled by a PhD student of the LIRIS laboratory.

Once we have the feature vectors and the classifier, we need a label for each feature vector indicating its class in order to realize a supervised learning.

On the supervised learning, the Machine Learning implements its algorithm to learn, using the feature vectors and the labels, to distinguish between the different classes. Then, a predictive model is created. When we want to classify a new feature vector, the model compares the values of the vector with the values that it has learned, and it assigns a label to the new vector. (cf. Figure 3)

![Figure 3. Machine Learning operation scheme](image)

Before the training starts, a projection of the feature vectors is done in order to convert the most correlated samples into uncorrelated samples. As a result, the feature vectors dimension is reduced and the final samples are more differentiate between them. That makes the classifier to be more discriminative when he is predicting a new class. The used technique for that is the PCA (Principal Component Analysis), which projects the feature vectors through an orthogonal transformation.
2.2. Feature Extraction

Given an already trained classifier, a Microsoft Kinect camera is used to provide the necessary feature vectors to predict the gesture to a remote server by applying a machine learning method. VOIR system is located in that server, and it recollects the information obtained in the camera in order to make all the required computations to predict the gesture realized by the user.

Kinect camera is characterized for providing 5 types of independent streams: color, depth, skeleton, interaction, and audio. In the already existing solution, VOIR uses the skeleton stream in order to obtain the feature vectors that will be used for train and predict the classifier.

The camera extracts information about the human skeleton; in fact, 20 recognized human joints positions are provided (cf. figure 4). Of these 20 joints, only 11 joints of them, corresponding to the upper body, are exploited.

For each joint, its state is also provided; such that three type of states are possible: “TRACKED” when the joint is fully visible, “INFERRED” when the joint is not fully visible and is based on the positions of the neighboring joint, and “NOT TRACKED” when it is not possible to follow the trajectory of the joint.

Also, a coordinate \( x, y \) and \( z \) is provided for each joint in order to situate its position.

![Figure 4. Skeleton joints detected by Kinect camera](image-url)
The procedure to construct the feature vector starts normalizing all the joint coordinates by the distance between HipCenter and ShoulderCenter. Because of that, we can work with the skeleton of every person, independently of its height, viewpoint and person morphology.

After normalizing, distances between all joints are computed. So, we have $11 \times 10 = 110$ distances between joints.

Next, three sets of angles are computed.

First set ($\alpha$) is composed by 9 angles that are obtained of the inclination between real and virtual bones.

Second set ($\beta$) is composed by 9 angles that are computed by the inclination between the projections of the first bone vector and and the vector $y$ (cf. Figure 5) on the plane perpendicular to the orientation of the second bone.

Third set ($\gamma$) contains 11 angles. Each angle is the inclination between each vector that connect a joint with the camera sensor position and the vector $z$, that, as we can see in the figure, points toward the camera.

So, finally our pose descriptor is composed by 9 $\alpha$ angles, 9 $\beta$ angles, 11 $\gamma$ angles, and 110 distances between joints. That makes our descriptor to have 139 dimension.

Also, we use our pose descriptors mirrored horizontally respect to the camera. So, finally, our pose descriptor has $2 \times 139 = 278$ dimension.
In order to have information about the temporal changing of the movements, the descriptors of three sequential frames are concatenated. Obtaining, then, the feature vector, that has $278 \times 3$ dimension. The spacing used between consecutive frames is of $\Delta t = 5$ frames. (cf. Figure 6)

Figure 6. Use of sequential pose descriptors in order to form the feature vector (In that case $\Delta t = 1$ frame)
2.3. Classifier – Neural Network

The classifier used for recognizing different types of gestures is the Artificial Neural Network (ANN).

An ANN is composed of units called neurons. Each neuron receives a set of inputs through interconnections and emits an output. This output is given by three functions (cf. Figure 7):

1. A propagation function, that is the sum of each input multiplied by its interconnection weight.
2. A transfer function, that modifies the output of the previous function. It can not exist, so its output would be the same as the propagation function’s output.
3. An activation function, that is applied to the transfer function’s output. It is used to define the neuron’s input and generally is given by the interpretation we want to give to such outputs.

We want our output labels to be “0” if the gesture is not recognized and “1” if the gesture is recognized. So, in our case the activation function used is the sigmoid function because the obtained values are in the interval \([0,1]\).

![Figure 7. Artificial Neural Network](image-url)
2.4. Training & Prediction Method

In VOIR system, two classifiers are trained:
- Filter neural network (Classifier 1)
- General neural network (Classifier 2)

The Filter neural network allows to determine if a new feature vector is or not a gesture. If it is, the General neural network is applied.

The General neural network decides which of the 8 possible gestures corresponds to the new feature vector. (cf. Figure 8)

![Diagram of VOIR system classifier structure](image)

Figure 8. VOIR system classifier structure

After knowing how the existing solution works, it only lacks to mention which kind of human gestures the VOIR system is trained to detect. These gestures are the 8 following ones:

- Stop
- Scroll: left-to-right
- Scroll: right-to-left
- Scroll: up-to-down
- Scroll: down-to-up
- Phone: pick up
- Phone: hang up
- Hi
2.5. System Architecture

The Kinect data stream, which we extract the training data afterwards, is generated by the Kinect camera. After that, it is sent to the client module using an API that specifies how the camera and the client should interact with each other. That API is called Kinect SDK.

The client module can run on the laptop of the robot or other remote devices over WIFI like distant computers and Android tablets.

The client provides an interface that the user can easily use for utilize the following applications:

- Choose the desired detection algorithm (object detection, face detection, gesture detection...).
- Choose the network connection where the desired image server is located.

The gesture recognition (and all other kind of recognitions) is computed in an image server because a set of algorithms and processes have to be computed in order to perform the image processing. For that, the client has to send the Kinect camera data to the server.

After the computations are done, the classification result is sent to the client. Then, it carries out the pertinent actions, like the robot movement.

Also, the image server records the computed training descriptors in a data server. This stored data will be used afterwards in classifier training sessions.

The image server and the client are connected by a network connection that carries a custom protocol.

Figure 9. System Architecture
3. Project development:

3.1. CONTRIBUTION

I have worked on the skeleton gesture recognizer and I started out from two already implemented programs: Neural Network Trainer and VOIR.

- Neural Network Trainer:

The first program is a neural network gesture classifier trainer. The classifier is implemented with OpenCV neural networks [1]. The training is supervised, and the labelling is done manually after obtaining the training data. Therefore, it is an offline training.

For each frame, the training data collected and used for training is:
- Position of different skeleton joints that provides Kinect.
- Angles between skeleton joints.
- Distances between skeleton joints.

Finally, for each training data vector, three consecutive frames are provided in order to have temporal information.

- VOIR:

The second program is the system VOIR, that it is already explained. It accesses the Kinect camera to collect the gestures data, and uses it to feed the trained classifier obtained in the previous program. Then, it displays the detected gesture for each frame.

The problem is that the classifier is not very reliable and does not give very good results. So, as it is said before, the proposed solution consists in completing the following tasks:
- Implementation of an online training system.
- Usage of new training data and new training vectors.
- Obtaining information from the finger gestures in order to reduce variability.

Due to the amount of labor of the first task and the adaptation between my system and the old system, I have worked only in the first and second task and the Hand tracking and Detection of finger gestures has not been realized finally. In the next section I explain how each one of the two tasks has been implemented.
3.1.1. Online Training

The idea is to improve the VOIR program by training the gestures that we have qualified as misclassified while the program is running. So, when you see that a displayed detected gesture is misclassified, you can train this specific gesture in order to improve its recognition performance.

First of all, we have to indicate which hand is going to be used for realize the pertinent gestures in the whole session. For that, we have to press one of the three following keys:
- “R”: Right hand is used for perform the gestures.
- “L”: Left hand is used for perform the gestures.
- “B”: Both hands are used for perform the gestures.

After that, we can collect data of a desired class everytime we want by pressing the corresponding button on the computer keyboard. The buttons for collect each class data are:

- “1”: Stop
- “2”: Scroll: left-to-right
- “3”: Scroll: right-to-left
- “4”: Scroll: up-to-down
- “5”: Scroll: down-to-up
- “6”: Phone: pick up
- “7”: Phone: hang up
- “8”: Hi

While the button is pressed, the data is stored for each frame and each training vector is automatically labeled as we indicate the label by pressing the button. So, this process is a ground truth collection because we gather the proper objective data for the training.

Once we have new data stored, we can update the neural network with a supervised learning as we have the labels corresponding to its respective feature vectors.

Taking into account that the new data is a ground truth set, we can not update the Filter neural network because all data recorded is labelled as a gesture. As we need gesture samples and no-gesture samples to train the Filter neural network, the only classifier that is trained is the General neural network.
3.1.2. New Training Descriptors & New Training Vectors:

- NEW TRAINING DESCRIPTORS:

In parallel to online learning, all captured data is recorded and stored (images, skeletons and ground truth), so that we can re-use it later.

Eventually, we use the recorded data in order to realize a batch training session, where all the recorded data is used for batch-training.

If we use this new training data besides the actual training data, the classification can be more accurate because we would have more information.

We store the training data during the session. A session starts when we run the VOIR program and it finishes when we stop the program.

This is the training data to be recorded in a session:

For each frame:

- A .txt file containing:
  - Status of each joint (INFERRED, TRACKED or NOT TRACKED).
  - Position of different skeleton joints.
  - Angles between skeleton joints.
  - Distances between skeleton joints.
  - Coordinates of the bounding box of the right hand.
  - Coordinates of the bounding box of the left hand.
  - Class name.
  - Class number.

Figure 1. Txt file structure for each frame, containing the frame data
- We store also an image of each hand. For that, we extract the bounding box around each hand (coordinates of the bounding boxes are store in the .txt file) and we crop the RGB Kinect image and the depth Kinect image in order to obtain, in each case, an image of the left hand and an image of the right hand. Finally, the images are binarized and stored in .PGM files (binary files) because their size are very small, and then the storing is conducted quickly.

So, we store 4 .PGM files:
- Depth right hand image.
- Depth left hand image.
- RGB right hand image.
- RGB left hand image.

The images are not used for training, but they are stored in case we want to revise them afterwards and do a manual labelling (as it is done in the existing solution).

For the whole session:

- A .txt file called "session.txt" contains information of all actions recorded during the session:
  - Beginning frame of each action.
  - Beginning timestamp of each action.
  - End frame of each action.
  - End timestamp of each action.
  - Class number of each action.
  - Class name of each action.
  - Hand that carried out each action (indicated by the keyboard buttons: L (Left) R (Right) and B (Both) before recording the data).

![Image of session.txt file structure](image-url)

Figure 12. "session.txt" file structure, containing information about all actions realized and recorded in the whole session
- Two .XML files, one for each hand, containing:
  - Elements called action that have two attributes: class and nr. class indicates the class number and nr indicates the action number.
  - For each element, we have childs corresponding to the frames of class number class. And each child has its frame number and the bounding box of the appropriate hand.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tags>
  <video>
    <session name/>
    <action nr="1" class="8">
      <bbox y="120" x="313" width="157" height="230" framenr="1"/>
      <bbox y="118" x="312" width="157" height="238" framenr="2"/>
      <bbox y="117" x="312" width="156" height="237" framenr="3"/>
      <bbox y="115" x="315" width="157" height="238" framenr="4"/>
      <bbox y="114" x="314" width="156" height="237" framenr="5"/>
      <bbox y="111" x="316" width="157" height="238" framenr="6"/>
      <bbox y="110" x="318" width="157" height="238" framenr="7"/>
      <bbox y="108" x="318" width="157" height="238" framenr="8"/>
      <bbox y="107" x="316" width="156" height="236" framenr="9"/>
      <bbox y="107" x="318" width="156" height="237" framenr="10"/>
      <bbox y="75" x="381" width="151" height="229" framenr="11"/>
      <bbox y="75" x="382" width="151" height="229" framenr="12"/>
      <bbox y="105" x="320" width="157" height="237" framenr="13"/>
      <bbox y="105" x="321" width="156" height="237" framenr="14"/>
      <bbox y="104" x="320" width="156" height="236" framenr="15"/>
      <bbox y="105" x="320" width="156" height="237" framenr="16"/>
      <bbox y="76" x="385" width="151" height="229" framenr="17"/>
      <bbox y="76" x="386" width="151" height="229" framenr="18"/>
      <bbox y="77" x="387" width="151" height="229" framenr="19"/>
      <bbox y="108" x="325" width="157" height="238" framenr="20"/>
      <bbox y="110" x="327" width="157" height="239" framenr="21"/>
      <bbox y="108" x="328" width="157" height="238" framenr="22"/>
      <bbox y="108" x="328" width="157" height="238" framenr="23"/>
      <bbox y="107" x="328" width="157" height="238" framenr="24"/>
      <bbox y="108" x="329" width="157" height="238" framenr="25"/>
      <bbox y="108" x="329" width="157" height="238" framenr="26"/>
      <bbox y="107" x="329" width="157" height="238" framenr="27"/>
      <bbox y="107" x="329" width="157" height="238" framenr="28"/>
      <bbox y="107" x="329" width="157" height="239" framenr="29"/>
      <bbox y="108" x="328" width="158" height="239" framenr="30"/>
      <bbox y="108" x="328" width="158" height="239" framenr="31"/>
      <bbox y="108" x="328" width="158" height="239" framenr="32"/>
      <bbox y="108" x="328" width="158" height="230" framenr="33"/>
    </action>
  </video>
</tags>
```

Figure 13. XML file structure
NEW TRAINING VECTORS:

Also, I have implemented a new structure of the training vectors in order to improve the classification performance. Instead of using the actual training vectors (three consecutive frames: 3P), we want to compute the following:

- current frame \( P(t) \).
- first derivative \( \delta P(t) \approx P(t+1) - P(t-1) \).
- second derivative \( \delta^2 P(t) \approx P(t+2) + P(t-2) - 2P(t) \).

The current frame, the first derivative and the second derivative are concatenated and compose the training vector structure: \( P \delta P \delta^2 P \).

If we do that, we will have much more temporal information about the movement and the classification accuracy can improve.

For this, I have relied on [3] (In section 3 we can see the structure of the training vector; and in section 5.3 we can see the benefits of using it).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Actions</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>joint angles</td>
<td>26.5</td>
<td>24.4</td>
</tr>
<tr>
<td>( P )</td>
<td>60.5</td>
<td>63.7</td>
</tr>
<tr>
<td>( 3P )</td>
<td>71.1</td>
<td>62.5</td>
</tr>
<tr>
<td>( 5P )</td>
<td>73.5</td>
<td>61.8</td>
</tr>
<tr>
<td>( 6P )</td>
<td>77.0</td>
<td>51.3</td>
</tr>
<tr>
<td>( \delta^2 P )</td>
<td>74.6</td>
<td>46.3</td>
</tr>
<tr>
<td>( P \delta P )</td>
<td>84.5</td>
<td>67.5</td>
</tr>
<tr>
<td>( P \delta^2 P )</td>
<td>89.7</td>
<td>68.1</td>
</tr>
<tr>
<td>( P \delta P \delta^2 P + conf. )</td>
<td>91.7</td>
<td>69.3</td>
</tr>
<tr>
<td>( P \delta P \delta^2 P + conf. + temp. )</td>
<td><strong>91.7</strong></td>
<td><strong>73.8</strong></td>
</tr>
</tbody>
</table>

Figure 14. Accuracy for each training vector type.
(The third one is the actual training vector, the eighth one is the training vector to be implemented)

While the existing training vector gives us information about the skeleton joint positions over the time, the new structure of feature vector gives us also kinematic information.

Taking the pose descriptor of frame in time \( t \), \( \delta P(t) \) is the difference between the position of the previous frame and the next frame; that gives us information about the position changing speed. So, knowledge of the velocity of movement is added.

\( \delta^2 P(t) \) gives us information about the change in directions and in velocity over time. Then, information about movement acceleration is also added.

Taking into account that we add kinematic information to the already existing static information, the movements of gesture are defined in a more precisely way. That means that the classifier is able to differentiate better between similar poses that have different directions of movements and different speed of performance.
3.2. BLOCK DIAGRAM OF THE SYSTEM

The final system is composed by 3 modules and has the following block diagram:
- **MODULE 1: Online Recognition “VOIR”**

**Input:** - General Neural network Model  
- Filter Neural network Model  

**Output:** - Updated General Neural network Model file  

“VOIR” module is the improvement of the second already implemented program (VOIR), mentioned in 3.1. Contribution section. It runs one of the three following processes, depending on the pressed button:

1) **Gesture Detection**

If no button is pressed, it extracts the feature vector information from the frames obtained from the Kinect camera. Then, the feature vectors are passed through the Filter classifier as an input, and it returns as a result:
   - 0 (the sample is negative, so it is not an action).
   - 1 (the sample is positive, so it is an action).

If the result of the previous classifier is 1 (positive sample), the feature vectors are passed through the General classifier as an input, and it returns as a result the name of the predicted class.

2) **Collect Data**

If any button between “1”-“8” is pressed (After choosing the hand that we are going to use for collect the data; buttons “L”, “R” or “B”), the program collect new data of the chosen gesture.

From the tracked skeleton, the color Kinect image and the depth Kinect image, it collects the necessary data (described in 3.1.2.) in order to update the neural network. Also, the data is stored for being used afterwards in an offline learning.

3) **Update Neural Network**

If “T” button is pressed and new data has been recorded, the General neural network weights are updated using the feature vectors recorded from the last time the neural network was updated.

Then the General neural network used from now on, while the program is running, is the updated one.
- MODULE 2: YML files creation module “xml2yml”

**Input:** - Collected data recorded in Module 1, specifically .txt files and .xml files.
  .pgm files are not used for training.
  - Already existing data: .txt files and .xml files.

**Output:** - Two .yml files containing a matrix each one. The first .yml file contains the feature vectors matrix and the second one contains the labels matrix.

“xml2yml” module is a new program added to the already existing solution. It reads all the collected data of all sessions located in the desired folder.

For each session it extracts the feature vectors and the labels by the following way:
  - Reading the XML files, when a new action element is found, its class number is taken and used for construct the label vector. (cf. Figure 1)
  - The feature vectors are constructed with the skeleton joints contained in the appropriate .txt files. These .txt files that are used are the ones with the frame number of each child contained in the corresponding action element.

All feature vectors and label vectors are pushed into several matrices(one for features and another for labels) that are stored in two different .yml files.

This module is capable to extract the feature vectors and labels of the new collected data, that is labeled online, and of the already existing data, that is labeled manually.
- **MODULE 3: Neural Network Training module “ann_train”**

**Input:** - .yml files containing feature vectors and labels obtained in Module 2.

**Output:** - General Neural network Model  
  - Filter Neural network Model  
  Also, .txt files containing the evaluation results are stored:  
    - *ann_train_trte.txt:* contains the evaluation results of mode *trte*  
    - *ann_train_cv.txt:* contains the evaluation results of mode *cv*.

“*ann_train*” module is the improvement of the first already implemented program (Neural Network Trainer), mentioned in 3.1.**Contribution** section. It trains the General Neural Network and the Filter Neural Network using the feature vectors and labels extracted from the .yml files created in the “xml2yml” module.

It can works in two modes, depending on the user selection:

1) **Mode trte (Train and Test)**

It trains the general neural network using a train dataset of the input feature vectors and labels. And a test is done afterwards using a test dataset of the input feature vectors and labels.

The partition between the train dataset and the test dataset is selected by the user, who indicates the train dataset percentage desired.

After this partition is done, the general neural network is trained using the first partition, and the test is done using the second partition.

In the 4.1.**Test Methods** section, *Train and Test* method is explained with more detail.

The performance results are stored in the *ann_train_trte.txt* file. This method is applied to the General Neural Network and to the Filter Neural Network, so a model for each neural network is created and stored in two .yml files. These .yml files are saved in the same place as the .txt file that contains the computation results.

Finally, it must be pointed that there is a special situation: when the selected train partition is equivalent to 100% of the whole dataset. In that case, the neural networks are trained using the whole dataset and no evaluation computation is done.
2) Mode cv (Cross-Validation)

This mode corresponds to cross-validation and it trains different General Neural Networks by appropriately partitioning the whole feature vectors dataset and the whole labels dataset. In each training, a cross-validation evaluation is performed, which is explained with more detailed in 4.1.Test Methods section.

The performance results are stored in ann_train_cv.txt. In this mode no neural network is stored as this mode is only used for performance testing.
4. **Tests and Results**

4.1. TEST METHODS

The tests that I have realized are done in order to differentiate the performance between the existing feature vectors (that only have static information) and the new structured feature vectors (that have static and kinematic information).

For that purpose, two type of tests have been done:
- Train and Test
- Cross-Validation

Both of the methods are implemented in the “ann_train” module as it is said in the previous section.

**TRAIN AND TEST:**

Taking the whole set of feature vectors and labels, it is partitioned into the train set and the test set.

In our case, I have assigned the train dataset as the 70% of the whole dataset. Therefore the test dataset corresponds to the rest of the whole dataset, its 30%.

After this partition is done, the neural network is trained using the first partition, and the test is done using the second partition.

For the test, the test samples are predicted using the already trained neural network, and the classification results are compared with the test labels.

If the classification result and the label are the same, the classification is correct.
Conversely, the classification is wrong.

After counting the number of correct classifications and wrong classifications, we can compute the Error Rate and the Accuracy:

\[
\text{Error Rate} = \frac{\text{Number of wrong classifications}}{\text{Number of total classifications}}
\]

\[
\text{Accuracy} = 1 - \text{Error Rate}
\]
CROSS-VALIDATION:

First of all, the whole dataset is divided into a selected number of folds; taking into account that every fold has the same size. The number of neural networks trained is the corresponding to the number of folds of the whole dataset. For each iteration, a neural network is trained; so, the number of iterations is also equivalent to the number of folds.

In each iteration, the following process is realized:
- All folds except one are used as the train dataset, and the other fold is used as the test dataset (so, in each iteration the test dataset fold is different) (cf. Figure 16). We train the neural network using the positive train samples, ergo the samples corresponding to an action.
- We divide the test dataset in the negative test dataset (samples not corresponding to any action) and the positive test dataset (samples corresponding to an action).
- A prediction is done using the negative test dataset as an input for the already trained neural network. Each classification result is compared with its corresponding test label. If they are equal, the classification result is true negative (\(tn\)); if not, the classification result is false negative (\(fn\)).
- A second prediction is done, but this time using the positive test dataset as an input for the already trained neural network. Each classification result is compared with its corresponding test label. If they are equal, the classification result is true positive (\(tp\)); if not, the classification result is false positive (\(fp\)).

After counting the number of \(tn\), \(fn\), \(tp\), and \(fp\), we can calculate the Precision and the Recall of this iteration:

\[
\text{Precision} = \frac{tp}{tp + fp} \quad \text{Recall} = \frac{tp}{tp + fn}
\]

After repeating this process for each iteration, we do an average of Precision and Recall of all iterations.
4.2. TEST RESULTS

In order to compare the existing feature vectors (3P) performance with the new structured feature vectors (P δP δ²P) performance, a “Train & Test” and a “Cross-Validation” tests have been done for each type of vectors.

In both cases, the collected data used for testing is the same one: a dataset of 30 sessions already labeled, that means an approximately number of 3800 feature vectors in each case. It can be considered a sufficient amount in order to have reliable results as the classifier can do an adequate discrimination between classes.

The tests have been performed for the Filter neural network and for the General neural network in order to have more comparison results. For the same reason, I have worked with different PCA dimensions and different number of neural network hidden units.

Two parameters have been fixed:
- In “Train & Test”, the train dataset comprise the 70% of the whole dataset and the test dataset comprise the remaining 30% of the whole dataset.
- In “Cross-Validation”, the number of folds is 5 for all the realized tests.

So, these are the different results for the different parameters, methods and feature vectors:

- **Train & Test:**

<table>
<thead>
<tr>
<th>General neural network</th>
<th>PCA dim = 160</th>
<th>PCA dim = 200</th>
<th>PCA dim = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>- ExistingVectors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hidden Units = 36</td>
<td>0.673781</td>
<td>0.67622</td>
<td>0.673171</td>
</tr>
<tr>
<td>Error Rate</td>
<td>0.326219</td>
<td>0.32378</td>
<td>0.326829</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General neural network</th>
<th>PCA dim = 160</th>
<th>PCA dim = 200</th>
<th>PCA dim = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>- New Vectors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hidden Units = 36</td>
<td>0.653571</td>
<td>0.65</td>
<td>0.635714</td>
</tr>
<tr>
<td>Error Rate</td>
<td>0.346429</td>
<td>0.35</td>
<td>0.364286</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. General neural network - Train & Test
<table>
<thead>
<tr>
<th>Filter neural network</th>
<th>PCA dim = 60</th>
<th>PCA dim = 100</th>
<th>PCA dim = 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Existing Vectors</td>
<td>Hidden Units = 40</td>
<td>Hidden Units = 65</td>
<td>Hidden Units = 80</td>
</tr>
<tr>
<td>Error Rate</td>
<td>0.496341</td>
<td>0.540854</td>
<td>0.546341</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.503659</td>
<td>0.459146</td>
<td>0.453659</td>
</tr>
<tr>
<td>Filter neural network</td>
<td>PCA dim = 60</td>
<td>PCA dim = 100</td>
<td>PCA dim = 150</td>
</tr>
<tr>
<td>- New Vectors</td>
<td>Hidden Units = 40</td>
<td>Hidden Units = 65</td>
<td>Hidden Units = 80</td>
</tr>
<tr>
<td>Error Rate</td>
<td>0.475968</td>
<td>0.469048</td>
<td>0.540476</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.524032</td>
<td>0.530952</td>
<td>0.459524</td>
</tr>
</tbody>
</table>

Table 2. Filter neural network - Train & Test - Cross-Validation:

<table>
<thead>
<tr>
<th>General neural network</th>
<th>PCA dim = 160</th>
<th>PCA dim = 200</th>
<th>PCA dim = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Existing Vectors</td>
<td>Hidden Units = 36</td>
<td>Hidden Units = 60</td>
<td>Hidden Units = 40</td>
</tr>
<tr>
<td>Precision</td>
<td>0.854161</td>
<td>0.760222</td>
<td>0.826961</td>
</tr>
<tr>
<td>Recall</td>
<td>0.471475</td>
<td>0.497709</td>
<td>0.462994</td>
</tr>
<tr>
<td>General neural network</td>
<td>PCA dim = 160</td>
<td>PCA dim = 200</td>
<td>PCA dim = 100</td>
</tr>
<tr>
<td>- New Vectors</td>
<td>Hidden Units = 36</td>
<td>Hidden Units = 60</td>
<td>Hidden Units = 40</td>
</tr>
<tr>
<td>Precision</td>
<td>0.887833</td>
<td>0.838245</td>
<td>0.87346</td>
</tr>
<tr>
<td>Recall</td>
<td>0.479256</td>
<td>0.465507</td>
<td>0.476422</td>
</tr>
</tbody>
</table>

Table 3. General neural network - Cross-Validation
As we can see in the previous tables, the new vectors improve the performance in both test experiments for both neural networks. The average improvement for each performance parameter is:

- General neural network Error Rate: 4,3%         Filter neural network Error Rate: 6%
- General neural network Accuracy: 8,3%           Filter neural network Accuracy: 6,6%
- General neural network Precision: 6,5%           Filter neural network Precision: 7%
- General neural network Recall: -1,9%               Filter neural network Recall: 3,4%

So, in almost all the experiments, there is an improvement of the performance when the new type of vectors is used. But the improvement is not as high as in Figure 14, where the accuracy performance augments in a 26% for actions and in a 8% for activities.

In our case, the accuracy for actions improves an 8,3% for the general neural network and a 6,6% for the filter neural network.

The improvement is not as high as we were expecting, but at least it exists and the classifications will be more accurate if the new structure of the vectors is used from now on.
5. **Budget**

The robot used for the project is an Eddie mobile robot manufactured by Parallax; and a Microsoft Kinect Camera has been used for record the gesture information. Microsoft Visual Studio 2010 is the software employed for implement the programs.

The involved staff is compound by a junior engineer and two supervisor engineers.

The estimated cost of hardware, software and staff is as following:

<table>
<thead>
<tr>
<th>Materials</th>
<th>Price/Unit (€)</th>
<th>Units</th>
<th>Total (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eddie mobile robot</td>
<td>1300</td>
<td>1</td>
<td>1300</td>
</tr>
<tr>
<td>Microsoft Kinect Camera</td>
<td>150</td>
<td>1</td>
<td>150</td>
</tr>
<tr>
<td>Microsoft Visual Studio 2010</td>
<td>1250</td>
<td>1</td>
<td>1250</td>
</tr>
<tr>
<td>Total Cost</td>
<td>-</td>
<td>-</td>
<td>2700</td>
</tr>
</tbody>
</table>

Table 5: Material cost

<table>
<thead>
<tr>
<th>Staff</th>
<th>Workers</th>
<th>Price/Hour (€)</th>
<th>Hours/Week</th>
<th>Monthly Salary (€)</th>
<th>Number of Months</th>
<th>Total (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Junior Engineer</td>
<td>1</td>
<td>8</td>
<td>35</td>
<td>1120</td>
<td>5</td>
<td>5600</td>
</tr>
<tr>
<td>Engineer</td>
<td>2</td>
<td>20</td>
<td>4</td>
<td>640</td>
<td>5</td>
<td>3200</td>
</tr>
<tr>
<td>Total Cost</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8800</td>
</tr>
</tbody>
</table>

Table 6: Staff cost
<table>
<thead>
<tr>
<th>Cost Type</th>
<th>Price (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material Cost</td>
<td>2700</td>
</tr>
<tr>
<td>Staff Cost</td>
<td>8800</td>
</tr>
<tr>
<td>Total Cost</td>
<td>11500</td>
</tr>
</tbody>
</table>

Table 7: Project total cost
6. **Conclusions and future development:**

An improvement of the gesture recognizer performance is possible if the new feature vector structure is used. It is like that because it offers kinematic information of the actions apart from the static information that skeleton joints gives us.

Regarding the new data collected, clearly the obtainment of action labels in real time is the most important and useful. Thanks to this new information, afterward neural network trainings can be realized without necessity of doing an exhaustive manual labeling using external tools. Also, and more important, it enables us to update the neural network in real time when we want to improve the recognition of a misclassified gesture.

Despite the amelioration on the gesture recognizer efficiency, more labors should be done in order to improve it more. For example, the detection of finger gestures that finally couldn't be realized would help in this task.
Bibliography:


Appendix:

A poster of the project has been realized for the LIRIS laboratory. It is an external attached document.