Personalized Motion Sensor Driven Gesture Recognition in the FIWARE Cloud Platform

Alexandros Preventis, Kostas Stravoskoufos, Stelios Sotiriadis, Euripides G.M. Petrakis
Intelligent Systems Laboratory
Department of Electronic and Computer Engineering
Technical University of Crete (TUC)
Chania, Greece, GR-73100
Email: {apreventis, kgstravo, s.sotiriadis, petrakis}@intelligence.tuc.gr

***Abstract*—Gesture recognition technology enables new means of user communication and interaction with machines. This work focuses on gesture recognition by analyzing data, obtained by motion sensors, in the cloud. We present *Interact*, a cloud based gesture recognition system that uses FIWARE. To promote the development of Future Internet (FI) applications, all functionalities of *Interact* are offered as cloud services, enabling elasticity, lower cost of maintenance and off-site efficient data storage. To promote productivity, *Interact* offers a REST API for recognizing, storing and managing customized gesture collections in the cloud as well as for subscribing to sensors. The system is sensor independent and is designed to be compliant with the most popular motion sensors (i.e., Leap Motion or Kinect). To demonstrate the functionalities of *Interact*, a system prototype has been developed utilizing the LEAP Motion sensor, offering all the previously described functionalities. The prototype is hosted on FIWARE Lab and is available for testing.**

***Keywords*-cloud application; future internet; gesture recognition; real-time translation; motion sensors; FIWARE;**

I. Introduction

Gestures have always been an integral part of human social interaction. They are typically used in real-life conversations to resolve or to express a meaning (for example raising a thumb up to signal that something is "ok"). Moreover there are communities of people with hearing or speech impairment that communicate explicitly through a set of gestures which form a "sign language". For those communities, communication with the rest of the world becomes a difficult task as most people are not familiar with sign languages. Also, the number of different sign languages (each country has its own) makes communication even more difficult.

Gesture recognition technology can address the problem of translating sign languages to text or speech and at the same time enable new means of interaction and communication with devices introducing a more natural way to control machines and enhance user experience [1]. Interaction by gesture recognition techniques is achieved by processing data from Web cameras, motion sensors or wearable devices like Myo[1]. Efforts towards this direction have produced a set of gesture recognition systems, some of which are discussed in Section II. However, there are several issues concerning current gesture recognition solutions:

[1]https://www.thalmic.com/en/myo

1) They are proprietary, featuring only a finite set of gestures that can be recognized, not allowing users to modify their gesture databases, create their own gestures or, personalize recognition of an existing gesture collection to their needs (e.g., hands anatomy and motion speed).
2) They are sensor-specific: Users cannot use the sensor of their preference, as current gesture recognition solutions are bound to specific sensors and can process data acquired by these sensors.
3) They cannot be accessed through the Web: They operate locally on the machine where the sensor is connected to.
4) They are developed and tested on personal computers, with limited computational resources [2]. Thus, there are limitations on the amounts of data or the number of users they can serve.

The work herein, which extends the one in [3], is trying to address the above issues by introducing a gesture recognition system in the cloud which analyzes data obtained by motion sensors. Motion sensors tend to become more and more popular as the concept of the Internet of Things (IoT) becomes a reality. The most popular motion sensors today are Microsoft Kinect[2], Asus Xtion Pro[3] both of which can track the whole human body and the Leap Motion[4] sensor which can track only the hands but with more accuracy. The output of those sensors are streams of data with vectors of features describing the human body with every change in position or posture.

We present *Interact*, a motion sensor driven gesture recognition system that operates in the Cloud and is realized by means of cloud services communicating with each other through REST APIs [4]. *Interact* can work with any motion sensor currently available and can support an infinite number of motion sensors operating concurrently.

Interact, has been designed as a cloud service in order to inherit its fundamental features. It includes various distributed services that belong to different providers and are installed in the FIWARE[5] platform. This model offers great benefits to the

[2]http://www.microsoft.com/en-us/kinectforwindows
[3]http://www.asus.com/Multimedia/Xtion_PRO
[4]https://www.leapmotion.com
[5]http://www.fiware.org

overall architecture of *Interact*. These are: (a) elasticity as the platform could allow various levels of resource provisioning, (b) there is no need for software updates and maintenance, (c) increase accessibility and collaboration in terms of availability of services to 3rd party users and developers, (d) centralized security offered by the FIWARE platform, (e) remote access from everywhere and anywhere through its powerful API that allows technology shift to seamless application development, and (f) customization and user tailored orientation through user personalized features (e.g., shared cloud storage collections for users).

The main functionalities of *Interact* are offered as cloud services to promote the development of Future Internet (FI) applications. Specifically we have developed open cloud services for:

- Recognizing both, static and motion gestures. In static gestures the hands do not move when the gesture is performed (like raising a thumb to signal ok) while, in motion gestures hands are moving like waving or swiping. Gesture recognition accuracy is enhanced by machine learning.
- Creating private gesture collections where developers can store gestures and use them to develop FI applications. *Interact* allows multiple users to subscribe on the cloud and receive or process the communicated hand gestures. Subscribed users in different places may take part in a lecture or develop an application that interprets the hand gestures.

To demonstrate the functionalities of *Interact*, a system prototype has been developed utilizing the LEAP Motion sensor. Two use cases have been developed:

1) A "gesture to speech" use case to demonstrate how *Interact* can be used to offer communication services for people with hearing or speech impairment. For this use case our system prototype features a gesture collection containing the American Sign Language (ASL)[6] finger-spelled alphabet and a gesture collection featuring some of the most commonly used gestures of everyday life (e.g., raising a thumb to signal "ok").
2) A "gesture to action" use case to show how gestures can be transformed to system actions and commands. In this use case we use *Interact* to (a) navigate through a web application, (b) to control a video game character and to (c) control a robot through the cloud. For each case we use a specialized gesture collection. The prototype is hosted on *FIWARE Lab*[7] and is available for testing upon request.

Background knowledge and related work are discussed in Section II. The architecture, gesture matching mechanism and use-case scenarios of *Interact* are discussed in Section III. Finally, in Section V, we state our conclusions and discuss future research issues.

[6]http://en.wikipedia.org/wiki/American_Sign_Language
[7]http://www.fiware.org/lab/

II. Background and related work

A. Recognition techniques

Hand gestures can be classified as static and dynamic. Static gestures can be recognized using classification or template matching. On the other hand, dynamic gestures are moving gestures and therefore require methods that can handle and understand motion. Mitra et al. [5] compare gesture recognition techniques with particular emphasis given to Hidden Markov Models (HMMs), particle filtering and condensation algorithm, finite-state machines (FSMs), and artificial neural networks (ANNs). Rautaray et al. [6] not only describe gesture recognition techniques, but also compare the various devices that are used for capturing gestures, including Microsoft Kinect, Nintendo Wii [8] etc.

B. Gesture recognition systems

Liang et al. [7] have developed a real-time gesture recognition system that recognizes one-handed gestures from the Taiwanese Sign Language (TSL). This system also uses HMMs and its average accuracy is 80.4%. The system is language-specific as it has been trained for recognizing the TSL. It also requires the user to wear a glove that transmits gesture data. Liu and Lovell [2] have developed a system for tracking real-time hand gestures captured by a Web camera and a standard personal computer with no specialized image processing hardware. Chen et al. [8] have also implemented HMMs for recognizing hand's postures. Rautaray et al. [9] have developed a system that employs computer vision algorithms and gesture recognition techniques which in turn results in developing a low-cost interface device for interacting with objects in virtual environment using hand gestures. The application uses a limited set of hand gestures defined within the application that are translated to specific commands. Bevilacqua et al. [10] have also developed a system based on HMMs for dynamic gesture recognition related to music. In their work they note that it is impractical to have a general gesture database as the gestures vary highly between performers.

C. FIWARE and Generic Enablers

FIWARE is open cloud-based infrastructure for cost-effective creation and delivery of Future Internet (FI) applications and services. It has been funded mainly the EU's Future Internet Public-Private Partnership programme, referred to as FI-PPP [9] for Internet-enabled innovation. FIWARE introduced an innovative infrastructure for cost-effective creation and delivery of services on the Web. It offers an open architecture and a reference implementation of a novel service infrastructure, building upon generic and reusable building blocks referred to as Generic Enablers (GEs).

GEs are modular cloud based components offering reusable and commonly shared functions, enabling the rapid development of Future Internet applications. The functionality and specification of GEs provided by FIWARE can be accessed

[8]http://www.nintendo.com
[9]http://www.fi-ppp.eu

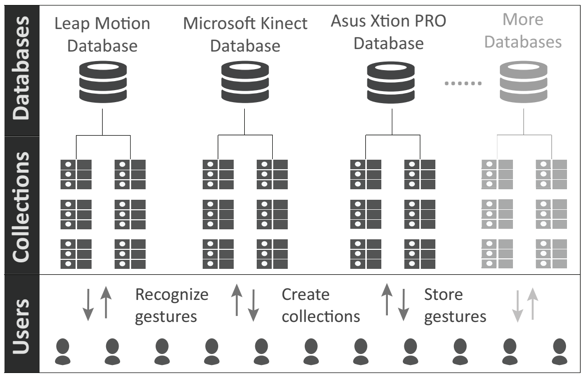

Fig. 1. Gesture databases and collections

through a public catalogue[10], in a way that developers can easily browse and select the appropriate APIs to use.

III. Interact

Interact is a cloud based gesture recognition system, based on the FIWARE platform, which promotes the development of motion sensor-oriented FI applications by offering its main functionalities as cloud services. *Interact* uses data obtained by motion sensors to recognize gestures. Motion sensors output streams of data with vectors of features that describe the human body. The format and schema of this data varies according to the motion sensor. Our system is designed to be sensor independent. Thus, it can operate with the most popular motion sensors currently available (namely the Microsoft Kinect, the ASUS Xtion PRO and the Leap Motion). To achieve sensor compatibility, *Interact* uses a data collector module (described in subsection III-A) that features sensor specific code in order to transform the data from each sensor to the desired format and schema. Considering this, to embed a new motion sensor, the data collector module must be modified accordingly. The prototype system that is currently available utilizes the Leap Motion sensor.

Gesture representation and organization is illustrated in Fig. 1. Gestures are organized into gesture collections which are part of sensor specific databases. Each database corresponds to a specific sensor and contains and all gesture collections in this database contain gestures that can be recognized only by using this sensor. For example, gestures in a collection created using the LEAP Motion sensor cannot be recognized by using a different sensor (e.g., Kinect). Gesture collections are distinguished into:

- Public gesture collections which are open and can be accessed by everyone.
- Private gesture collections which are available only to subscribers (access is granted by the creator of the collection).

In both cases, only the creator of a collection is permitted to add new gestures. Gestures are stored in the form of objects with key-value pairs that represent gesture features (e.g., number of fingers, rotation of hand etc.). When a gesture is captured, gesture data is obtained by the sensor and the system matches the input to the gesture records in the database of the specified sensor. Gesture representation and the matching proccess are thoroughly discussed in IV-A. *Interact* can process static and motion gestures. In static gestures the hands do not move when the gesture is performed (e.g., raising a thumb to signal “ok”) while, in motion gestures hands are moving (e.g., waving “hello”). Static gestures are represented by a set of features representing the hands while motion gestures are represented by an array containing a “snapshot” of features for every position in the path of the moving hands.

The functionality of *Interact* is offered as a service through a REST API that is described in Table I. Authenticated users can access resources by making HTTP requests (GET, POST or DELETE) to the service end-point. Consider the following example:
There is a user with username *user1* who has created a gesture collection named *Sign-Language* by using the *kinect* sensor. In order to access the collection and list all available gestures we should make a GET request to */Users/user1/dbs/kinect/Collections/Sign-Language/Gestures*. The response of this service call would be a JSON[11] string with all available gestures.

By using *Interact*’s API users can have access to the following functionalities:

- **Create and manage gesture collections:** The same gesture may have a different meaning in different applications. This way users can create application specific collections and populate them accordingly. Collections can be public (accessible by everyone) or private (available only to subscribers).
- **Store gestures on the cloud:** Users are allowed to append new gestures in collections they create. When a gesture is captured by the motion sensor, features are extracted and stored in JSON format. Every gesture is assigned a self-descriptive name and optionally some user-defined data. The data can be used like a stored procedure for rapid application development
- **Gesture recognition:** Users can recognize Gestures can be accessed by making HTTP requests to the *gesture collection*’s URI. After matching a gesture, a JSON object containing the name of the matched gesture is returned to the user.
- **Subscribe to gesture collections:** Users can subscribe to gesture collections and use them for building applications. A publish/subscribe service module (a module that informs context-subscribers about context updates) handles the subscriptions and notifies users about changes on gesture collections they are subscribed to (e.g., new gestures added).
- **Subscribe to user:** Users have the ability to broadcast sensor data real time. This way subscribers are updated with the recognized gestures real time (with the use of

[10]http://catalogue.fiware.org/

[11]http://json.org/

TABLE I
INTERACT REST API

Method	URI (http://base_URL/...)	Action
GET	/users/:username/dbs/:dbname/collections/:collectionname/gestures	Get all gestures in a collection
POST	/users/:username/dbs/:dbname/collections/:collectionname/gestures	Add new gesture in a collection
DELETE	/users/:username/dbs/:dbname/collections/:collectionname/gestures/:gestureid	Delete a gesture
GET	/users/:username/dbs/:dbname/collections	Get the gesture collections
POST	/users/:username/dbs/:dbname/collections	Create a collection
DELETE	/users/:username/dbs/:dbname/collections/:collectionname	Delete collection
GET	/users/:username/dbs/:dbname/collections/:collectionname/subscriptions	Get collection subscribers
POST	/users/:username/dbs/:dbname/collections/:collectionname/subscriptions	Subscribe to a collection (:uid)
DELETE	/users/:username/dbs/:dbname/collections/:collectionname/subscriptions/:userid	Unsubscribe from collection
POST	/users/:username/dbs/:dbname/collections/:collectionname/recognize	Recognize gesture
GET	/users/:username/subscriptions	Get the subscribers of the user
POST	/users/:username/subscriptions	Subscribe to user
DELETE	/users/:username/subscriptions	Unsubscribe from user

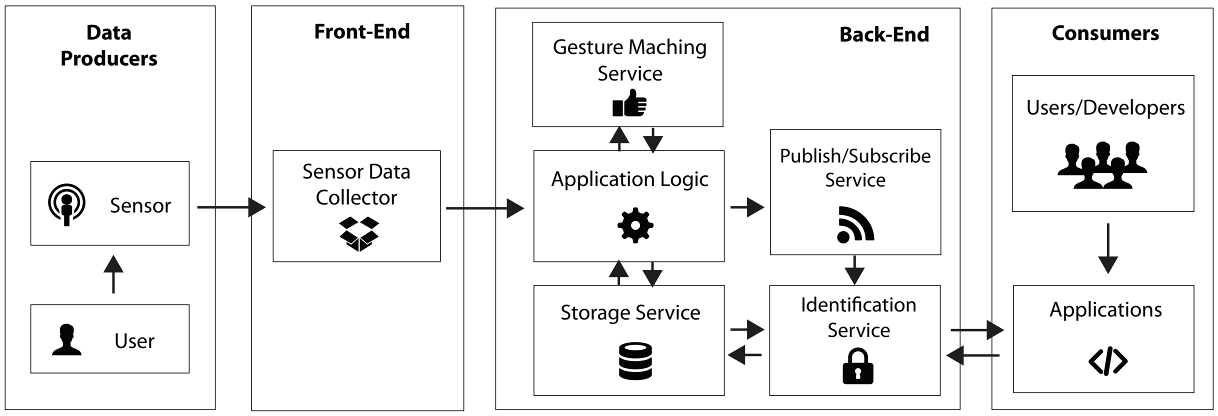

Fig. 2. Architecture of Interact.

the publish/subscribe service).

Interact supports multiple users, each one being able to utilize multiple motion sensors and create multiple gestures and gesture collections. Considering this, there are several reasons why *Interact* relies on the power of the cloud:

- The amount of streaming data that motion sensors produce and the fact that data must be processed real time call for computational power on demand.
- There is no limit to the amount of gestures or gesture collections that a user can create. Therefore, storage on demand is also a requirement.
- The number of users and sensors may vary significantly over time, either increasing or decreasing. Thus, an elastic environment that would allocate or free resources on demand is needed.

Considering the above, conventional physical computers are not capable of fulfilling the applications requirements for elasticity and scalability which are major keys to its success.

A. Architecture

Figure 2 illustrates the architecture of *Interact*. It consists of four basic blocks: (a) the producers being motion sensors who produce the data, (b) the front-end which is the service end-point, (c) the cloud back-end where the application logic is implemented and (d) the consumers-end where applications operate, accessing the services of *Interact*. Each of these blocks consists of one or more modules that provide certain functionality:

- **Sensor Data Collector:** This front-end module is responsible for collecting data from the motion sensor and converting it to JSON to be processed in the back-end.

JSON is a lightweight format for data interchanging, thus, it is ideal for representing and storing sensor data.

- **Storage Service:** The Storage service is responsible for data storage and retrieval.
- **Gesture Matching Service:** This module implements the gesture recognition process. It classifies gestures and when the user inputs a gesture for recognition it compares the gesture to the all available classes to find which class the gesture is more similar to.
- **Publish/Subscribe Service:** The role of this service is to inform subscribers about changes in a domain of interest (e.g., new gestures have been added in a collection).
- **Identification Service:** This module is used for user authentication and access authorization.
- **Application Logic:** This module encapsulates the application logic of *Interact*. It handles and serves the requests sent from the front-end or consumer applications.

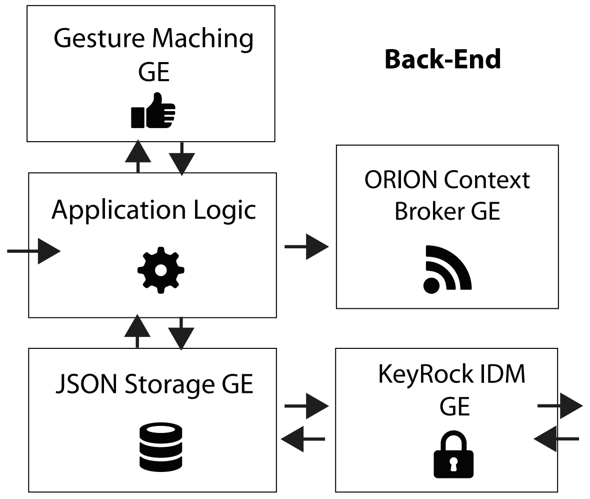

Fig. 3. Architecture of Back-End block. The service modules that are displayed in Figure 2 have been replaced with Generic Enablers.

IV. Implementation

The modules that compose the main functionalities of *Interact* are operating at the back-end. This includes four modules that provide discrete services. *Interact* is an FI application and complies with FIWARE specifications. In that end, the modules that provide these services are Generic Enablers. Figure 3 illustrates the architecture of the back-end, where the service modules that are displayed in Figure 2 have been replaced with Generic Enablers.

- **JSON Storage GE:** This GE provides NoSQL[12] database management services through a REST API. Its users can perform CRUD (Create-Read-Update-Delete) operations on resources by using the basic HTTP methods (POST, GET, PUT, DELETE). In addition, it allows users to query over the stored resources.
- **Gesture Matching GE:** This GE takes as an input the gesture data and the selected collection and responds with the class that the given gesture is most probable to belong to. The gesture matching process and algorithm are explained in subsection IV-A
- **ORION Context Broker GE:** The Orion Context Broker[13] provides a publish/subscribe mechanism. Using the Orion Context Broker, users can subscribe to context elements (e.g., a room whose temperature and atmospheric pressure are measured) and get updated on context changes. In addition, they can use predefined conditions (e.g., an interval of time has passed or the context element's attributes have changed) so they get context updates only when the condition is satisfied. This module allows users to subscribe to other users gesture collections to use them for building applications or to subscribe to a user who is broadcasting information.
- **KeyRock Identity Management GE:** The KeyRock GE[14] provides secure and private authentication from users to devices, networks and services, authorization and trust management, user profile management, privacy-preserving disposition of personal data, Single Sign-On (SSO) to service domains and Identity Federation towards applications. Identity Management is used for authorizing foreign services to access personal data stored in a secure environment.

[12]http://en.wikipedia.org/wiki/NoSQL
[13]http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[14]http://catalogue.fiware.org/enablers/identity-management-keyrock

A. Matching Gestures

The gesture recognition process of *Interact* is taking place in the Gesture Matching GE. The GE implements a Naive Bayes Classifier and provides a REST API for accessing its resources and operations.

A gesture object consists of several features, whose values are either produced by the sensor (e.g., hand position values), or are calculated by using the sensor-produced features (e.g., the distance between the hands is calculated by comparing hand position values). The classifier takes as input a gesture object (JSON) that represents the gesture and contains key-value pairs representing the features.

1) Feature Selection: The features that will contribute in the classification process are selected, from a given set, by the developer considering the needs of each application. For example, an application that just counts the fingers of each hand would not use all the available features, but just one or two. This set of available features contains all the gesture features made available by the Leap Motion sensor, plus some additional features calculated using the aforementioned. More specifically, each gesture is represented as a vector with the following attributes:

- *Hand Number:* An integer that describes the number of hands that participate in the gesture.
- *Hand Distance:* The distance of the hands (if both are used in the gesture).
- *Finger Number:* The number of fingers (from both hands) that participate in the gesture.

For each hand in the gesture, the features are:

- *Hand Number:* An integer that describes the number of hands participating in the gesture.
- *Finger Number:* An integer that describes the number of fingers participating in the gesture.
- *Hand Distance:* The distance of the hands (if more than one participate in the gesture).
- *Hand Direction:* A vector in the three dimensional space, pointing from the hand's palm toward the fingers.
- *Palm Normal:* A vector in the three dimensional space, perpendicular to the plane formed by the palm of the hand.
- *Palm Position:* A vector in the three dimensional space, representing the distance of the palm center and the motion sensor.
- *Sphere Center:* The center of a hypothetical sphere, fit to the curvature of the hand.
- *Sphere Radius:* The radius of the hypothetical sphere.
- *Pitch:* The rotation of the hand around the x-axis.
- *Roll:* The rotation of the hand around the z-axis.
- *Yaw:* The rotation of the hand around the y-axis.

And for each finger of each hand:

- *Length:* The length of the finger.
- *Direction:* A vector pointing outwards of the fingertip.
- *Finger Distance:* An array that contains the distances between each finger and all the other fingers of the hand.

2) Classification: In gesture recognition the classifier has to deal with missing features, since different gestures contain different number of features (i.e., a gesture that uses two fingers has less features than one that uses five). Moreover, in order to make *Interact* give real-time translation of gestures, the classification had to be instantaneous. For these reasons, the classifier implemented in this work is the Naive Bayes Classifier[15].

Each gesture that appears in the training set of the classifier has been labeled by the user (trainer) during the training process. The different labels are the classes used by the classifier. During the translation process, the classifier decides in which class the gesture belongs to and responds with the label of this class. This is achieved by calculating the posterior probability that this gestures comes from a specific class. For instance, given a gesture **x**, the posterior probability that this gesture is member of the class labeled as "A", is:

$$P(\mathbf{x}|A) = \frac{P(A) \times \prod_1^n P(F_i|A)}{p(F_1, \ldots, F_n)} \tag{1}$$

Where $F_i, \ldots, F_n$ are the feature variables that are present in each gesture, $P(A)$ is the prior probability that the gesture belongs to the class "A" and its value is given by the frequency of appearance of each class in the training set. The denominator in formula 1 is a scaling factor dependent only on the feature variables, and is a constant since the feature variables are known.

[15]http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Naive_Bayes_classifier.html

TABLE II
ACCURACY WHEN PERFORMING K-FOLD CROSS-VALIDATION.

Folds (k)	**Accuracy %**
2	68,32
5	71.10
10	71.93
20	73.08

The term $P(F_i|A)$ is the probability that the given attribute comes from the class "A" and, using a Gaussian distribution assumption, is given by:

$$P(F_i|A) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{-(F_i - \mu)^2}{2\sigma^2}\right) \tag{2}$$

Where σ^2 and μ is the variance and the mean value respectively, of the specific feature variable in the class "A".

These calculations are executed for each class C and the classifier's outcome will be the class for which the quantity $P(F_i|C)$ is maximized.

3) Testing: The system has been tested on the ASL finger-spelled alphabet. The alphabet consists of 26 static one-handed gestures, each one representing a letter of the English alphabet. The data that has been used as a training set for the classifier consists of $1,225$ characters taken from short documents found in the World Wide Web. This way, the relative frequency of each letter contained in the training set approaches the relative frequency of this letter in the American Language[16].

The classifier has been validated using K-Fold Cross-validation. During this process the training set is randomly partitioned into k equal size subsets. Of the k subsets, a single subset is retained as the validation data for testing the model, and the remaining $k-1$ subsets are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsets used exactly once as the validation data. The k results from the folds are then averaged to produce a single estimation. Table II demonstrates the changes in the classifier's accuracy when performing K-Fold cross-validation, for various values of k. Notice that the accuracy reaches its highest values for $k \geq 10$. For $k = 10$, the size of each fold is 123 gestures, thus the training set size is $1,102$ gestures. The predicted accuracy, based on the training set only, for this value of k is $71,93\%$. In conclusion, in order to achieve maximum accuracy in the translation, the classifier has to be trained with a data set of more than $1,000$ gestures.

4) Performance: The performance of *Interact* is affected by three main factors:

- Network delays: *Interact* is based on a Service Oriented Architecture (SOA). This means that for any action that is performed by the user, one or more components communicate with others via methods of the HTTP Protocol, in order to fulfill it. The delays introduced by these HTTP

[16]http://en.wikipedia.org/wiki/Letter_frequency

TABLE III
TIME (MILLISECONDS) FOR ACTIONS TO COMPLETE.

Action	Collection100	Collection10
Creation	1258 ms	432 ms
Creation (Server)	1242 ms	420 ms
Recognition	91 ms	86 ms
Recognition (Server)	17 ms	11 ms

requests and responses are the main delays of the system and they appear to be the bottleneck of the responsiveness of *Interact*.

- Classifier creation: When the collection is selected to be used for gesture recognition, the classifier for the gestures has to be created from scratch. This is the most time consuming process of *Interact* but it does not affect the responsiveness as it is executed when the collection is selected (before the recognition starts).
- Gesture recognition: Since the classifier is already created before the recognition process, the recognition time itself is minor. The biggest portion of the delay of the recognition is added by the network delays.

Table III demonstrates the time (in milliseconds) that each action takes. *Collection100* is a gesture collection that contains 100 different gestures and similarly *Collection10* contains 10 different gestures. The creation of the classifier when Collection100 is selected takes an average of 1258 ms. The average End-to-end time[17] for recognition of a gesture for the same collection is 91 ms when the server only needs 17 ms to complete the recognition. The rest 74 ms are delays added by the network and the browser to show the results. The fluctuations of times when Collection10 is selected are slightly decreased (as the comparisons that the classifier has to make are significantly less) but in general follow the same pattern with Collection100.

B. Use Case Scenarios

In this section we present two use cases that demonstrate the functionality of *Interact* and prove our system useful in real-life scenarios. The first scenario demonstrates a "gesture to speech" translation service and the second "gesture to action" service.

1) Gesture to Speech: In this scenario people with hearing or speech impairment seek information from a public information spot. There, with the use of *Interact* the system is able to translate hand gestures into speech and enable communication. The scenario would involve:

1) A motion sensor to obtain gesture data.
2) *A*ccessing an instance of Interact to transform the gesture to text.
3) A gesture collection with the sign language.
4) A "text to speech" service to transform the output of *Interact* to speech.

Our system prototype currently features a "gesture to voice" functionality, as described above, using a small gesture collection with common everyday gestures and Google's text to speech service [18].

2) Gesture to action: Transforming gestures to action commands would enable developers to:

(a) Create a whole new User Interface (UI) experience for users by creating applications controlled explicitly with gestures (e.g., controlling a smart TV and changing channels by swiping a hand left or right). (b) Create motion sensor driven video games. Game moves can be mapped to actions that correspond to gestures (e.g., a boxing punch or a sword slash). (c) Map actions to robot executable commands and control robots remotely through the cloud (e.g., control the production line of a factory).

The process is the same in each case and it would require:

1) A motion sensor to obtain gesture data.
2) *Interact* to transform the gesture to an action command.
3) A gesture collection with action commands that apply to the use case.
4) A specific module to transform the output of *Interact* to (a) application commands, (b) robot executable commands or (c) game character movement commands.

In our system prototype we are using *Interact* to control the Sphero[19] robotic ball. We are using two gesture collections for this purpose. The first one is for moving the ball around. This collection features four gestures: Left, right, up and down. For example, when we use the gesture "Left", the ball moves left. The second is for changing the color of the ball according to the number of fingers we use in a gesture. This collection features five gestures, the numbers one to five which correspond to five different colors.

V. CONCLUSIONS AND FUTURE WORK

In this work we present *Interact*, a cloud based gesture recognition solution that offers it's functionality as a service. *Interact* is sensor-independent, can be accessed through the Web and allows users to create their own gestures and gesture collections. This gives *Interact* considerable advantages over current gesture recognition systems. A prototype system, hosted on *FIWARE Lab* and utilizing the Leap Motion sensor has been built and is available for testing.

The proposed solution is expected to create new market opportunities for SMEs to develop FI applications and commercialize innovative products and services. Moreover, as a gesture recognition system, *Interact*, highlights and promotes new means of communication for people with hearing or speech impairment making their interaction with other people easier. Considering the above, potential markets that could benefit from our solution are health-care provision, monitoring

[17]End-to-end time refers to the temporal duration from the instant that the gesture was captured at the Client, to the instant that the Client receives the translation of the gesture.

[18]http://translate.google.com/

[19]http://www.gosphero.com/

and surveillance, educational training and the video game industry.

As future work we intend to focus on the following issues: First, to embed the Microsoft Kinect and the Asus Xtion PRO to our prototype. This will allow our system to recognize more complex gestures, featuring the whole human body, which could offer future market openings for monitoring, training, video game development etc. Second we intend to improve our gesture recognition system by evaluating different machine learning algorithms. A potentially good choice that could increase our recognition accuracy (especially considering motion gestures) is the use of Hidden Markov Models as they can handle the temporal dimension of dynamic gestures.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no 604691 (project FI-STAR).

REFERENCES

[1] V. I. Pavlovic, R. Sharma, and T. S. Huang, "Visual interpretation of hand gestures for human-computer interaction: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 677–695, Jul. 1997. [Online]. Available: http://dx.doi.org/10.1109/34.598226

[2] N. Liu and B. C. Lovell, "Mmx-accelerated real-time hand tracking system," in *IVCNZ 2001*, 2001, pp. 381–385.

[3] A. Preventis, K. Stravoskoufos, S. Sotiriadis, and E. G. Petrakis, "Interact: Gesture recognition in the cloud," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, Dec 2014, pp. 501–502.

[4] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[5] S. Mitra and T. Acharya, "Gesture recognition: A survey," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 3, pp. 311–324, May 2007.

[6] S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial Intelligence Review*, pp. 1–54, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10462-012-9356-9

[7] R.-H. Liang and M. Ouhyoung, "A real-time continuous gesture recognition system for sign language," in *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, Apr 1998, pp. 558–567.

[8] F.-S. Chen, C.-M. Fu, and C.-L. Huang, "Hand gesture recognition using a real-time tracking method and hidden markov models," *Image and Vision Computing*, vol. 21, no. 8, pp. 745 – 758, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885603000702

[9] S. S. Rautaray and A. Agrawal, "Real time hand gesture recognition system for dynamic applications," *Int J UbiComp*, vol. 3, no. 1, pp. 21–31, 2012.

[10] F. Bevilacqua, B. Zamborlin, A. Sypniewski, N. Schnell, F. Gudy, and N. Rasamimanana, "Continuous realtime gesture following and recognition," in *Gesture in Embodied Communication and Human-Computer Interaction*, ser. Lecture Notes in Computer Science, S. Kopp and I. Wachsmuth, Eds. Springer Berlin Heidelberg, 2010, vol. 5934, pp. 73–84. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12553-9_7