# The Intelligent Bartender: Project Report

Sverrir Thorgeirsson, Pablo Garcia-Nieto Rodriguez, Ismael Casabán

Dep. of Information Technology, Uppsala University

May 31, 2016

**Summary**

We chose to implement Project 1.4: Gesture Recognition. We implemented algorithms and trained a neural network to recognize eight different hand gestures to be performed using a Leap Motion controller which detects hand movements. We collected Leap Motion recordings of friends and fellow students performing the gestures, which we then used to generated training data consisting of feature vectors of the pertinent aspects of the hand movements. Every algorithm we implemented and later compared was very successful at distinguishing between the eight gestures. We then used those algorithms as the backbone of a program which a person can use to order drinks and food. The program takes the gestures as input, decodes their meaning and communicates the results to the user through a user interface which uses both sound and visuals. That way, the program is able to have a dialog with the user, which works successfully as we verified through user testing.

The work between three project participants was distributed equally with everyone contributing in some way to every component of the project, including the data collection, the blog and the presentations for the feedback sessions. That being said, each of us focused on different parts:

Pablo wrote various bits of useful code, most importantly the user interface and code that gathered the output from the Leap Motion controller. He also wrote code that streamlined the data collection process.

Sverrir wrote and tested most of the machine learning algorithms and the cross-validation of them using the training data, and algorithms for detecting if the hand is still or moving. He also wrote much of the report.

Ismael visualized the training data and created a JSON parser for the input from the Leap Motion device. He also helped with choosing the feature vectors and deciding the gestures.

## 1 Project description

Our project is called The Intelligent Bartender[1] and consists of software in Python that enables a user to order items from a bar using only hand gestures, which is especially convenient for visually-impaired users or those who are hard-of-hearing. The user can select between eight hand gestures to interact with the software, including two

[1]During the project implementation, we kept a blog illustrating our process. Here is a link: `www.intelligentbartender.blogspot.com`

| | |
|---|---|
| `circle` | Order pasta as a meal |
| `rock` | Order hamburger as a meal |
| `pistol` | Order cola |
| `scissors` | Order cocktail |
| `pinky` | Pay with cash |
| `come` | Pay with card |
| `roll` | Cancel the last selection |
| `stop` | Confirm the order (repeated two times) |

Figure 1: The eight gestures and their meaning in alphabetical order. Note that it is only required to use one hand for any gesture.

gestures for canceling their last action (if their gesture was misinterpreted or the user changed their mind) and for confirming their order. The user is expected to perform their gesture using a Leap Motion controller, which is further explained in Section 2. Our software then uses a machine learning algorithm to understand which gesture was performed (see Section 3), having learned to distinguish the gestures after being trained on data generated by eight users of the system.

### 1.1 Gestures

The user is expected to initially place their hand horizontally above the Leap Motion controller with their fingers in a relaxed position. Then after a small pause, they should perform any of the eight gestures, then keep their hand still for about half a second until the software notifies the user via audio and visual output that their gesture has been registered. In Figure 1, the different gestures are named and their meaning explained (for diagrams, see Figures 5 and 6 in Appendix A). Note that the user has the freedom to use either their left or right hand.

## 2 Feature selection and data acquisition

The Leap Motion controller is a device intended for tracking hand and finger movements, using cameras that generate between 60 and 200 frames per second of data [2]. The data collected in each frame includes the positions of the user's palms and hand bones, the yaw, roll and pitch of their wrists and the direction of their finger bones. In order to allow a machine learning algorithm to distinguish between our eight hand gestures, we decided that using seven measurements from the Leap Motion controller should be sufficient: the yaw and pitch of the wrist (measured in degrees) and the distance from the end of the distal bone of each finger from the center of the palm.[2] The feature vector of a given data sample consists of the difference between each of those measurements when the hand was in its initial position and when the hand had finished performing the hand gesture.

We collected data from eight people to use for training and testing of our algorithms. Three of those people included us, the project developers. The data providers varied in age, gender, hand size and dexterity. Each person was asked to perform each

[2]Furthermore, we also make a distinction between the left and the right hand of the user for the sign of the rotation of the wrist (for example, rotating the left hand clockwise gives the same result as rotating the right hand counter-clockwise).

of the eight gestures five times with each hand. This gave us a total of 640 data points. During this process, we noticed that the Leap Motion controller is an imperfect device that can give invalid output under certain circumstances, such as when the user performs the hand gesture too fast or without much care. Therefore we were required to do some manual preprocessing of the data by monitoring if the Leap Motion visualizer showed roughly the same visual output as was happening in reality. However, we tried to keep this preprocessing to a minimum in order to allow a machine learning algorithm to recognize systematic hardware errors.

After the data collection was finished, we plotted the data in two dimensions using t-SNE dimensionality reduction as taught during the course (see Figure 2). Most of the data appeared to be separated into eight clusters, making it promising that machine learning algorithms would work wel.

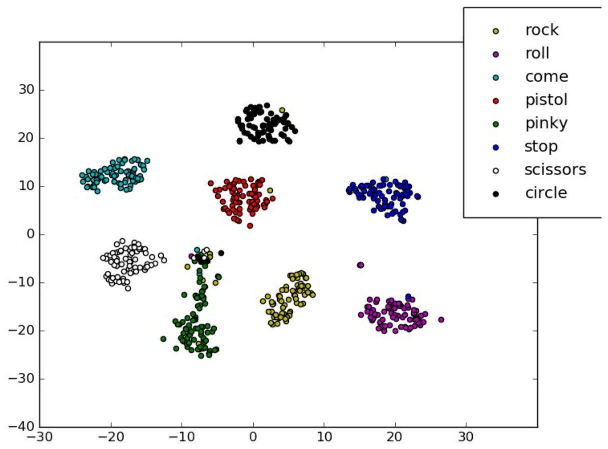


Figure 2: Visualization of the training data using t-SNE.

## 3 Machine learning methods

We implemented three different supervised learning classification methods to learn from the training data of 640 labeled data points. The goal was to enable our system to classify data generated by ordinary users of our system, i.e. to infer the correct hand gesture of the user. In this section, we describe the algorithms we implemented, then use leave-one-user-out (LOUO) cross-validation[3] to i) find the algorithm's optimal hyperparameters ii) compare the effectiveness of the algorithms on our classification problem.

### 3.1 k-nearest neighbors and weighted k-nearest neighbors

Considering that the data we collected seemed to be rather well-clustered (see Figure 2), we expected the simple k-nearest neighbor (KNN) algorithm as taught during the course to give acceptable results. The KNN classifier has the advantage of being a

[3]This means that for each user X that provided testing data, we use the data generated by all users *except* X as training data and then use X's data to test the algorithm. The final error rate we use is the average error of each user's testing data.

| User no. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | average |
|---|---|---|---|---|---|---|---|---|---|
| Error w. 64 neurons | 3.75% | 7.50% | 13.75% | 5.00% | 2.50% | 5.00% | 1.25% | 0.00% | 4.84% |
| Error w. 10 neurons | 1.25% | 7.50% | 16.25% | 5.00% | 13.75% | 5.00% | 3.75% | 0.00% | 6.56% |

Figure 3: The classification error after testing the verification dataset of each user using LOUO cross-validation.

simple algorithm to implement which can nevertheless give impressive results. It does nevertheless have some drawbacks. First of all, the method does not discriminate between any of the $k$ neighbors of the object which is to be classified. For example, the $k$th neighbor of the object has the same impact on the classification as the object's nearest neighbor, which is unnatural; one would like to weight the closest neighbors of an object more heavily than the ones which are further away. Secondly, it is not clear how to deal with ties using the KNN algorithm, i.e. when two or more classes could be chosen

In order to solve those issues, we implemented the weighted k-nearest neighbor (WKNN) classifier as described by Dudani [1]. The difference between WKNN and KNN is that in WKNN, the impact of each neighbor is weighted according to the function

$$f(x_0, x_i, k) = \begin{cases} 1, & \text{if } i = 1 \\ \frac{d(x_0,x_k)-d(x_0,x_i)}{d(x_0,x_k)-d(x_0,x_i)}, & \text{otherwise} \end{cases}$$

where $d(x, y)$ is the distance between data points $x$ and $y$, $x_0$ is the point to be classified and $x_i$ is its $i$th nearest neighbor.

We used the cross-validation scheme as described above in order to determine the optimal value of the hyperparameter $k$ for both the KNN and the WKNN classifier. As shown in Figure 4 in Appendix A, a relatively low value of $k = 5$ worked best for $KNN$ while a higher value such as $k = 20$ was one of the best choices for $WKNN$. When both algorithms were given the optimal value of their hyperparameter, the average error on the verification data sets was very similar, or between 4 and 5%.

## 3.2 Artificial neural network

In addition to the above algorithms, we trained an feed-forward artificial neural network (ANN) from the `pyBrain` module to learn from the data. Unlike the above algorithms, we now had a large number of hyperparameters to choose from, but initially we settled for a neural network with a large number of hidden neurons (64) that used the BackPropagation algorithm with a momentum and a weight decay. For each user, we split the user-generated data into training and testing set (using the proportions 80-20) after holding out the dataset from a given user and then used the testing set for early stopping regularization. After that, we fed to the network each verification set generated by the user that was held out (see results in Figure 3) and got an average error of 4.84%. We also tried the same algorithm with a lower number of hidden neurons (10) but this gave worse results (see Figure 3).

## 3.3 Comparison

When performing LOUO cross-validation on each of the above algorithms, we get a similar error rate that ranges between 4.4% and 4.8%. Therefore we believe that there is not a meaningful difference between using the three algorithms given our data, which is not very surprising; as shown on Figure 2, most data points (around 90-95%) appear to belong to close clusters, meaning that the most of the data should be linearly separable. Meanwhile, a small portion of the data is outliers that would be hard to classify correctly using any algorithm.

## 4 Final version and evaluation

For the final version of our software, we designed a user interface (see Figure 7 in Appendix A) that highlights the options selected by the user, which are signaled using only hand gestures. We implemented an algorithm that detects if the hand is still or moving to see if the user has finalized their hand gesture. After the user has completed their gesture and hence with the software, it will display their action and its interpretation on the screen and through audio, thereby creating a small dialogue between the user and computer.

The software appears to work near-perfectly, no matter which machine learning algorithm we use to classify the data. When we use the program, the gestures are recognized correctly more than 90% of the time, and when there is a mistake, using the cancellation gesture should quickly solve the issue. There is a low risk of accidentally confirming your order through misinterpretation of your gesture because the confirmation gesture needs to be performed two times in a row.

In order to confirm this, we asked a few people in the hallway of the institution who were unacquainted with our project to try out our software. After explaining to them how our software and gestures worked, they were without error able to order the items that they wanted.

## 5 Ethical concerns

For several reasons, the bartender software could not be integrated as-is to an actual bar environment, at least not without human supervision. Most importantly, the software does not have any way to see if the user of the software is underage and cannot order alcohol, and secondly, it cannot tell if the user is too inebriated to consume more alcohol. Computer vision technologies could be implemented (with some difficulty) in order to answer those two questions, but they could never be perfectly accurate and could return errors that could be constructed as discrimination; for example it would not be ethical if the software tended to consider women more drunk than men, but errors like that could easily happen depending on how the algorithms were implemented. Furthermore, the software is not as well-equipped as humans to deal with issues like allergies and disabilities, which could potentially have disastrous consequences.

## 6 References

[1] Sahibsinghi A. Dudani (1976), The Distance-Weighted k-Nearest Neighbor Rule, *IEEE Transactions on Systems, Man, and Cybernetics* (SMC-6), 325-327.

[2] “Controller — Leap Motion JavaScript SDK v2.3 documentation”. Link: https://developer.leapmotion.com/documentation/javascript/api/Leap.Controller.html. Retrieved 2016-05-19.

## Appendix A: Tables and figures

| Value of $k$ | KNN success rate | WKNN success rate |
|---|---|---|
| 1 | 0.947 | 0.947 |
| 2 | 0.947 | 0.947 |
| 3 | 0.948 | 0.947 |
| 4 | 0.956 | 0.947 |
| 5 | 0.956 | 0.947 |
| 6 | 0.955 | 0.950 |
| 7 | 0.953 | 0.950 |
| 8 | 0.953 | 0.950 |
| 9 | 0.951 | 0.950 |
| 10 | 0.950 | 0.950 |
| 11 | 0.952 | 0.948 |
| 12 | 0.950 | 0.950 |
| 13 | 0.955 | 0.952 |
| 14 | 0.955 | 0.952 |
| 15 | 0.953 | 0.953 |
| 16 | 0.955 | 0.952 |
| 17 | 0.950 | 0.952 |
| 18 | 0.950 | 0.952 |
| 19 | 0.950 | 0.952 |
| 20 | 0.950 | 0.952 |
| 21 | 0.950 | 0.953 |
| 22 | 0.952 | 0.955 |
| 23 | 0.952 | 0.955 |
| 24 | 0.952 | 0.955 |
| 25 | 0.952 | 0.955 |
| 30 | 0.952 | 0.953 |
| 35 | 0.947 | 0.955 |
| 40 | 0.944 | 0.955 |
| 45 | 0.945 | 0.953 |
| 50 | 0.944 | 0.953 |
| 75 | 0.934 | 0.950 |
| 100 | 0.921 | 0.952 |
| 150 | 0.895 | 0.942 |
| 300 | 0.434 | 0.934 |
| 560 | 0.125 | 0.931 |

Figure 4: Values of the hyperparameter $k$ and how it affects the performance of the KNN and WKNN algorithms. Not surprisingly, for KNN it decreases sharply as $k$ increases past 50, but for WKNN it remains relatively constant independently of how $k$ is chosen.

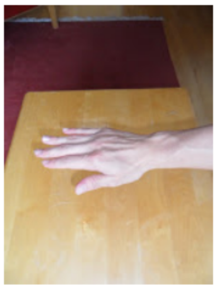
| *Picture* | *Gesture Description* |
|---|---|
|  | **Initial** All the gestures begins with the same static gesture. |

Figure 5: The initial hand gesture.

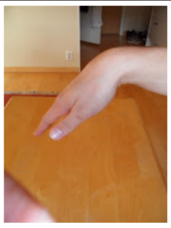
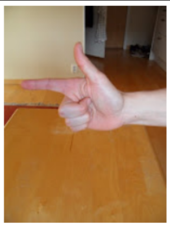
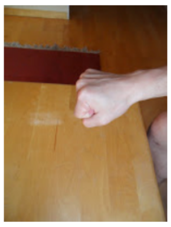
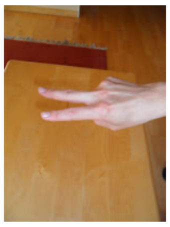
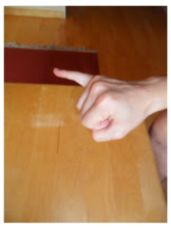
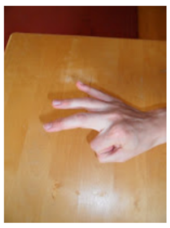
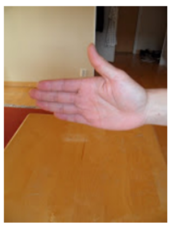
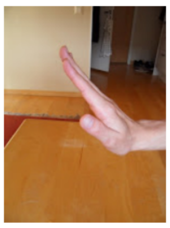
| *Picture* | *Gesture Description* | *Picture* | *Gesture Description* |
|---|---|---|---|
|  | *COME* |  | *PISTOL* |
|  | *ROCK* |  | *SCISSORS* |
|  | *PINKY* |  | *CIRCLE* |
|  | *ROLL* |  | *STOP* |

Figure 6: The eight hand gestures.

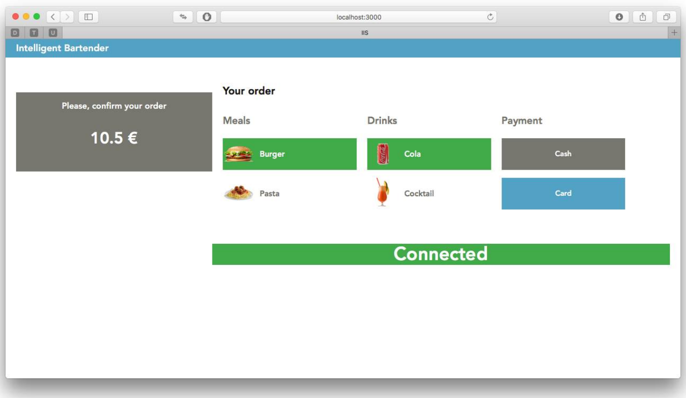


Figure 7: The user interface of the project, with the options that the user has selected colored in green.

## Appendix B: The specification

See next page. Note that the gestures we decided on ended up being different than defined in the specification after we became more familiar with the limitations and workings of the Leap Motion controller.

# Design Specifications

Intelligent Systems

Sverrir Porgeirsson, Ismael Casabán, Pablo García-Nieto

# Table of contents

# Project goals

The goal of this project is to provide the software for a bartender robot targeting customers with different disabilities or impairments (deaf, dumb people…) that make it difficult to order drinks or food in a bar. A good solution to tackle that issue is to implement intuitive **gesture recognition**, where users can complete the ordering process with no further complications.

The aim of the final prototype is to be a small independent module which can be placed in a bar and can assist users throughout the overall process, allowing them to order either alcoholic or non-alcoholic drinks, the meal of the day and their preferred payment method.

# Functional requirements

In this section, different functional requirements are stated divided by the category they belong to, according to the project goals and the description of the project provided for the course.

## Interface requirements

- Users can start the ordering process.
- Users can can choose one alcoholic or non-alcoholic drinks and the meal of the day.
- Users can choose whether to pay by card or cash.
- Users should know at any point the status of the ordering process.
- Users can cancel the order.
- Users should have feedback on whether a gesture was detected or not.

## System requirements

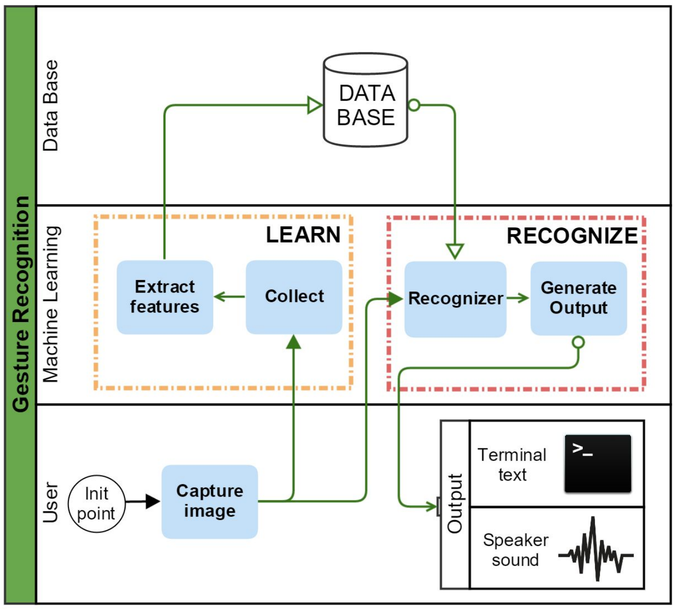
- The system can detect quickly eight different gestures.
- The system provides audio and text feedback.
- The system has a training mode.
- The system can manage card and cash payments.
- The system provides confirmation of the purchase.

## Security requirements

- The system handles payment information securely.

## System architecture

Our system is by necessity a simplification of the software for an actual bartender robot, since we can clearly not support the connection of the system to a payment system or a drink-making robot, for example. However, we provide the the rest of the system as described in the image below:

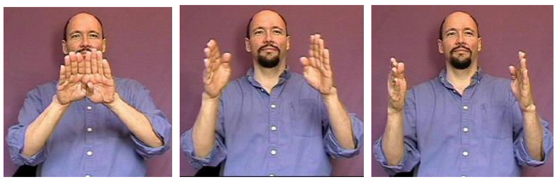
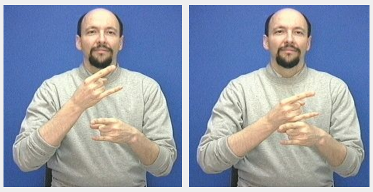
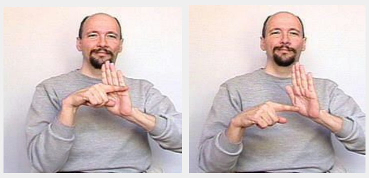
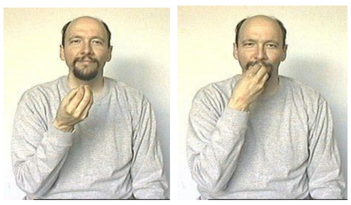
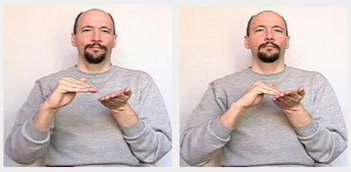
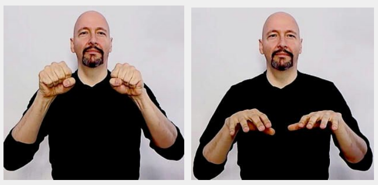


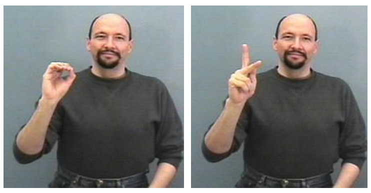
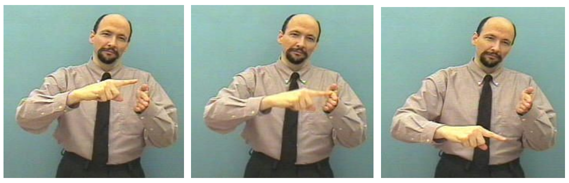
Note that the image shows an iterative process. The user will receive a notification from the system (a sound and text) that notifies them of the interpretation of their message, after which the user can send the next message. For example, if the interpretation of the gesture is wrong, the user can cancel the process and restart.

In order to aid the recognition of the images from the user, we intend to use a Leap Motion controller. We will generate data consisting of videos clips of the gestures we intend to use, which we will then use to train an artificial neural network so that it can recognize gestures in videos that it has not seen before.

## Control flow and interaction

The main purpose of the project is to replace spoken communication with gesture commands, but also providing a natural and intuitive conversation between the user and the bartender robot. That is why gestures are going to be intercepted all at once, making it possible for the users to configure the purchase details in their preferred order.

| Option | Gesture |
|---|---|
| Start the order process |  |
| Choose alcoholic drink |  |
| Choose non-alcoholic drink |  |
| Choose meal of the day |  |
| Choose pay by card |  |
| Choose pay by cash |  |

| | |
|---|---|
| Accept the order |  |
| Cancel the order |  |

Paired exclusive instructions have been highlighted in grey. When detected, the opposite pair will be avoided in the recognizement algorithm shrinking the amount of possible gestures.