AR-WIZARD

HAND GESTURE CLASSIFICATION IN AUGMENTED REALITY WITH LEAP MOTION

RASMUS B. HEMMINGSEN, au516408
ANDERS MEIDAHL, au513037
NICKLAS NIELSEN, au502453
JENS K. SCHRADER, au339396

ADVANCED AUGMENTED REALITY PROJECT

October 2019

Advisor: João Belo

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

AR-WIZARD

RASMUS B. HEMMINGSEN
ANDERS MEIDAHL
NICKLAS NIELSEN
JENS K. SCHRADER

Hand gesture classification in augmented reality with leap motion

Advanced Augmented Reality Project
Department of Computer Technology
Science & Technology
Aarhus University

October 2019

Augmented quotes.
You know you're in love when you can't fall asleep because augmented reality is finally better than your dreams.

— Dr. Seuss - Augmented

One person's craziness is another person's augmented reality.

— Tim Burton - Augmented

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AR	Augmented Reality
ASL	American Sign Language
BPP	Backpropagation Perceptron
CNN	Convolutional Neural Network
DNN	Deep Neural Network
HMD	Head Mounted Device
k-NN	*k*-Nearest Neighbour
ML	Machine Learning
MSE	Mean-Square-Error
MSEP	Mean-Square-Error Perceptron
NCC	Nearest Class Centroid
NN	Neural Network
NSCC	Nearest Subclass Centroid
OVO	One-Versus-One
OVR	One-Versus-Rest
RBF	Radial Basis Function
SVM	Support Vector Machine

1 INTRODUCTION

Augmented reality enables users to see and interact with objects and effects that are not present in the real world. Thus, augmented reality has the possibility to make a more magical world where anything is possible. AR-Wizard will use these possibilities of augmented reality to make the users feel, see and even manipulate the magic that we cannot see in the real world. Through AR-Wizard the users will become powerful wizards fighting off invading zombies with magical spells. The spells will be conjured by the user correctly performing different hand gestures. A visualization of AR-Wizard's original game idea is provided in figure 1.

Both augmented reality and machine learning are becoming more popular in the world as the potential in them is huge. This project's focus is to combine the two concepts and use them to investigate hypotheses 1 and 2. Different machine learning algorithms will be implemented, tested and compared with one another to validate the ease and accuracy of the methods.

Classifying a hand gesture is not possible from a single image as a gesture involves movement. Therefore, the hand gestures will be captured in a period of three seconds using an HMD. The HMD will provide the user with the feeling of a first-person game, thus a more real feeling that the user is the wizard fighting of zombies compare to just controlling a virtual figure. Multiple features will be extracted from the recording and processed through a machine learning algorithm to classify the recorded hand movement to a known hand gesture. The algorithm will provide a classification of the movement and a confidence level of that given classification. The confidence level will provide information regarding whether or not the user is performing the correct hand gesture to conjure a spell.

The HMD used in AR-Wizard consists of a Leap Motion[1] and an RGB-Camera. The Leap Motion will track the hand movement of the user and the features of the gesture will be extracted from the Leap Motion. The RGB-Camera will provide a visualization of the real world to which the spells and zombies will be placed into.

Hypothesis 1 *Augmented reality can effectively be used as a tool for comparing machine learning algorithms.*

Hypothesis 2 *Neural Network is the best performing machine learning algorithm for hand gesture recognition in a game using leap motion.*

1 https://www.leapmotion.com accessed 2019/05/12

Figure 1: Visualization of AR-Wizard original game idea.

2 RELATED WORK

This chapter surveys previous work related to gesture recognition with leap motion, machine learning algorithms, and virtual hand prediction.

There already exist multiple papers describing systems using Leap Motion with machine learning, [1, 2, 5, 8, 12–15]. The focus for multiple of these papers [2, 8, 14] is on recognizing sign language since this is a domain where it would help a lot of people if this could be automatically translatable. Most of the signs are static but some of them are dynamic and therefore an analysis of a single frame is not good enough. In the article describing American Sign Language (ASL)[8] a series of important features were extracted from the Leap Motion API and then some custom features were made from the extracted features. In this article, the position was completely ignored in the dataset following the logic that a sign should be able to be done at all positions. In this article, the methods used for classification were k-NN and SVM. Four years later in 2018, a similar dataset[2] was created again to be used for classification of ASL with SVM and DNN. The results for SVM were similar but the DNN was around 12% points better than the SVM.

The possibility to use both a Leap Motion and a Kinect device was described in an article [13] as a way to achieve very high accuracy in real-time. Here the combination of the Leap Motion API features together with the depth calculation from the Kinect was proposed.

Some of the differences between using an HMD and using a phone for Augmented Reality (AR) is described in an article [9] from 2014, where one of the highlights with using an HMD is the possibility for the user to use the hands. One of the cons of the HMD compared to the mobile is the decrease in mobility.

In order to detect what direction a user is pointing some calculations are needed, some work in this area has been done. Examples of this is where the user's physical hand was extended/replaced with a virtual hand so objects out of reach could be interacted with[6].

Game special effects are increasing demand for digital entertainment and the performance of the game scene and the special effects become more and more close to reality. This is described the article [16] from 2017, where there is a big focus on particles in Unity[1].

1 https://unity.com/ accessed 2019/05/14

3 ANALYSIS

The following chapter describes the methods and important frameworks from the related work, which are relevant for the project, and how it helps solve the hypothesis.

Creating an AR world can be achieved in many different ways. This can even be achieved using only a smartphone, where the camera record the world, analyzes and calculate the coordinate in the room and from there the user is able to place virtual objects in the room. Though using the phone requires the user to hold their phone in front of their eyes with one hand. Therefore, an analysis of what experience the user should have when playing AR-Wizard was made. As AR-Wizard is all about recognizing hand gestures, using one hand to hold a smartphone did not seem appropriate. This also removed the first-person view of the world, thus making the feeling of being the character less real. To provide the user with first-person view of the world, the possibilities of HMD was researched. Here the HoloLens[1] were looked at, which enabled for the recorded data to be displayed in front of the user's eyes, thus making the experience near life-like. Though the technology to track the hands had to be developed. This could be prevented by using the HoloLens 2 as it has build in feature to track the user's hands. Though the HoloLens 2 was only just released on the 24th of February. The HoloLens even though it is a great candidate for this project was discarded because of the limited access the university could provide to such a device. As such an alternative was made with the Leap Motion and an RGB-Camera mounted on a cap. This alternative approach provided the visualization as if it was first-person, though not visible in front of the user's eyes, as the visualization is on a computer screen. This allowed the user not to have the hands occupied holding devices but moved the visualization away from the user, thus decreasing the experience of being the character. More details about the HMD can be seen in chapter 5.4. The Leap Motion provides the tracking of the user's hand and makes it possible to extract lots of information regarding the coordinates and finger position and much more. Using the Leap Motion for extracting features comes with a lot of benefits compared to using only the RGB-Camera. One benefit being that no extra image processing was required as the data provided by the Leap Motion gave information about all the joints in the hand in the form of coordinates. Furthermore, this also meant that no Convolutional Neural Network (CNN) was required which again provided a lot of benefits. Removing the

1 https://www.microsoft.com/en-us/hololens accessed 2019/05/12

CNN from the project means that there was no need for training this part which might be error prone. Also if a CNN were used to extract the features of the hands, lots of data would be required where different skin colors would be used and in a lot of different lighting conditions. All of this is abstracted away as the Leap Motion provides this directly.

A model for image processing is no longer required due to the Leap Motion, but a model for the estimation of which hand gesture is performed is still required. For this multiple different machine learning algorithms were investigated: Nearest Class Centroid (NCC), Nearest Subclass Centroid (NSCC), k-NN, SVM, BPP, MSEP, and DNN. The last three models are Neural networks which were expected to perform the best [2]. All of these models were implemented and tested to evaluate the performances of the models in comparison to one another.

Because the use case in this project is to conjure spells from hand gestures it was desired to make custom hand gestures. This was desired because sign language or other hand gestures have not been able to summon spells previously in other applications. Therefore making new hand gestures to conjure spells seemed appropriate for the project. Being able to record and extract features from the custom hand gestures to train the Machine Learning algorithms were implemented into the solution. The recorded data were then labelled for the algorithms as supervised Machine Learning algorithms were used.

The development program Unity was used for the game itself. Unity was part of the learning course prior to the current course for this project which made it seem appropriate to use for the game development. Unity provides a lot of tools for game development in such that effects and virtual objects can be manipulated. This is the basis of the AR-Wizard world. Furthermore, Unity integrates with Vuforia[2] which enables images targets inside the AR. For the game itself, an image target was used to spawn zombies for the wizard to shoot and this image target enabled for an anchor point in the world which could be used for throwing the spells out into the world.

2 https://www.ptc.com/en/products/augmented-reality accessed 2019/05/17

4 DESIGN AND METHODOLOGY

In this chapter, the architecture of the system and API will be explained as well the gesture design choices. Furthermore, the design of different machine learning algorithms are explained. Afterward a section about the design in AR and in particular projectile trajectory, occlusion, world anchoring, and scaling. Then at the end of the chapter, there is a conclusion on the design.

4.1 ARCHITECTURE

The flow of creating and shooting spells is illustrated in the flowchart in figure 2. The API is abstracted out of this flowchart but is explain in details in figure 3 instead. The threshold is a parameter that can be tweaked to optimize the model to get less false positive. The first time the "Record 3 sec of data" step records for three seconds. Afterwards it continuously remove and adds one second of recording every second. This allows for a classification every second instead of every third second.

As figure 2 shows the program does nothing if there is no hand registered. The flow is constantly running and therefore to make less stress on the program and the API the hand constraint is added to minimize the traffic between the program and API.

Figure 3 shows a more detailed flow of the API when it receives data. It must be said that the preparation of the machine learning models is not in this flowchart as it is only a onetime operation. The API requires two parameters; Machine Learning (ML) method name and data. If the method does not exist an error is returned.

4.2 HARDWARE

AR-Wizard will use a laptop to run the game and the Machine Learning API. The video will be captured with a webcam and shown on the laptop in Unity and the hand gestures will be captured by a Leap Motion.

4.2.1 *Leap Motion*

Leap Motion is a small USB peripheral device designed to be used on a desktop or on an HMD. The device consists of three infrared LEDs and two cameras, each tracking in the 850 nanometers spectrum. The data received is then streamed through USB and then the software

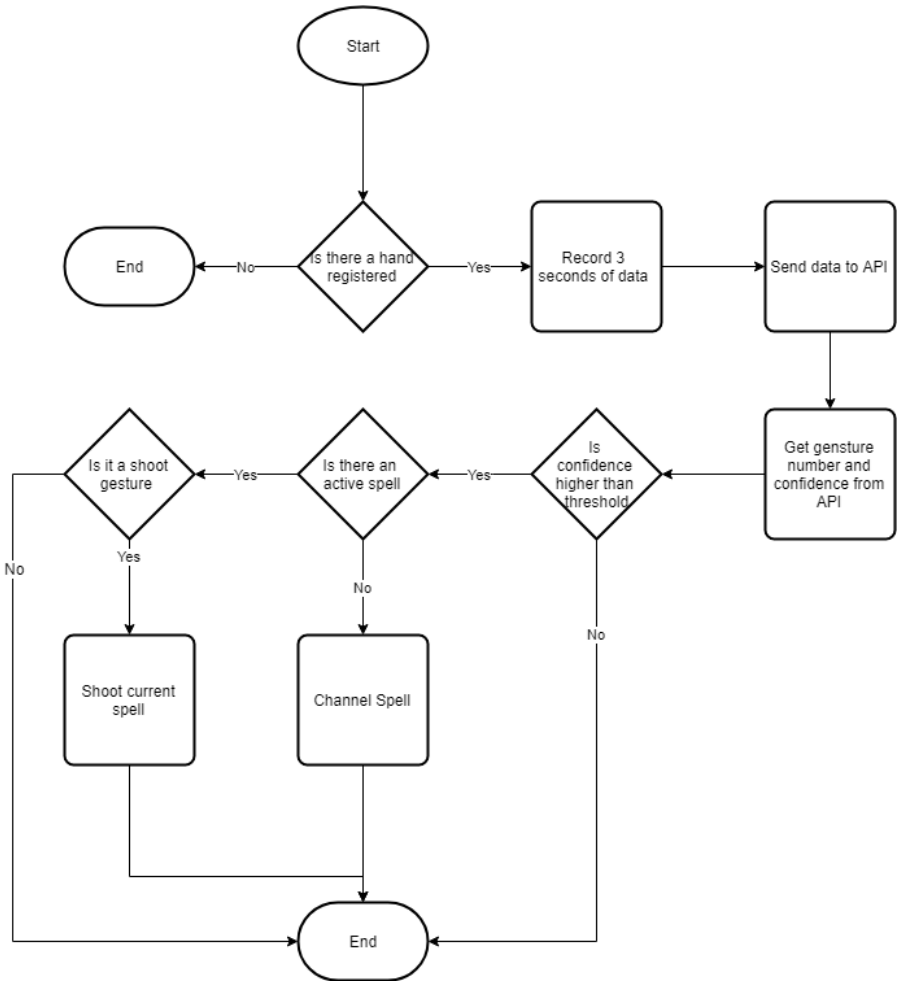

Figure 2: Flowchart for shoot and spell creation.

uses advanced algorithms on the raw data to reconstruct a 3D representation. The 3D representation is then used to construct the full hand by inferring the position of occluded objects. This data is then exposed in the Leap Motion API in Unity, where the different objects like fingers and palms are available as objects[3].

In this project, the Leap Motion is used for generating the feature vector which should be used for the classification. This process is described in section 4.3. The Leap Motion was chosen since it gives precise hand position data even in different lighting conditions and differences in skin color. This would allow the machine learning algorithms to just receive a feature vector with hand data and not an image, thereby there was no need for a CNN to generate feature vectors.

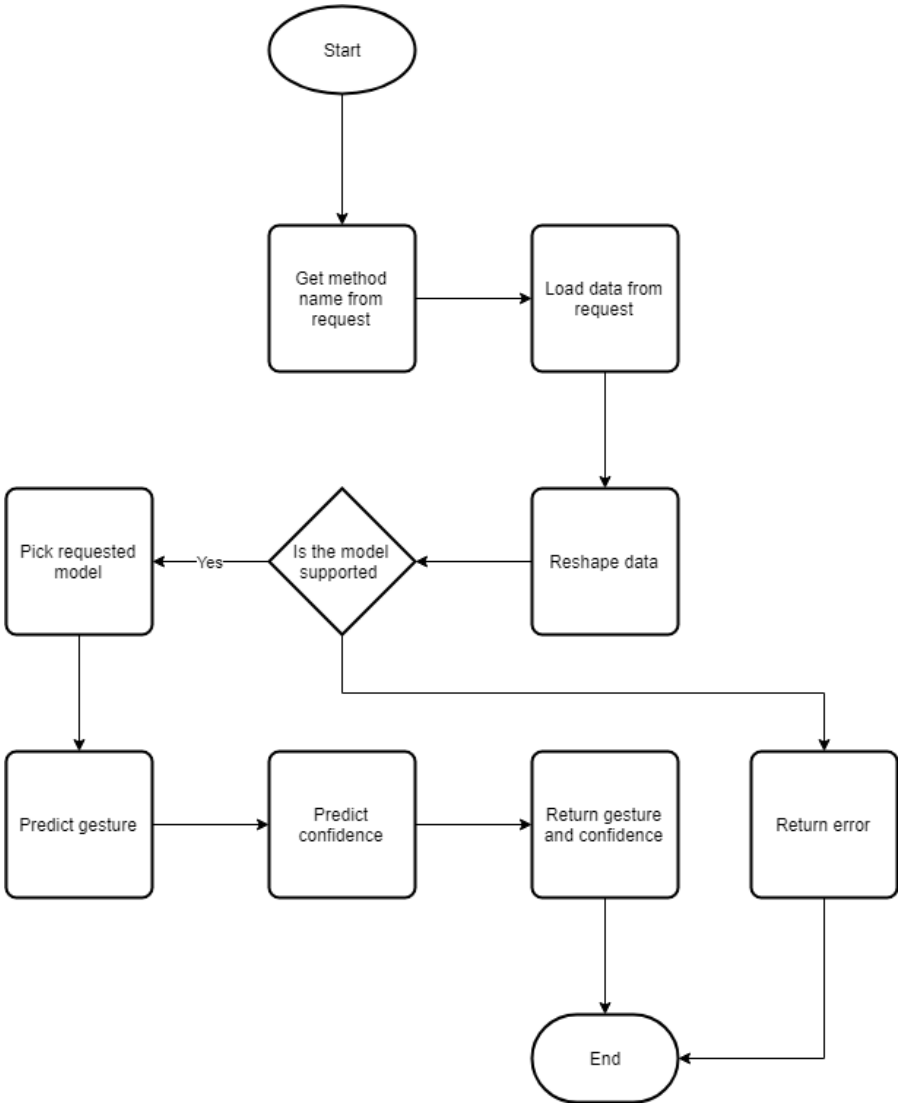

Figure 3: Flowchart for API.

4.2.2 *Webcam*

It was in the analysis decided to not use an HMD and therefore a webcam should be used to enable the user to see the AR on a screen. This is not the optimal experience for an AR game, but with the time limit of the project and the limited access to HMDs this was a trade-off done for this project. The webcam has a resolution of 1080p.

4.3 GESTURE CLASSIFICATION

The feature vector to use for classification is extracted from the Leap Motion API. Here features deemed important are inserted into a feature vector. The features selected were initially inspired by two papers [2, 5].

The features chosen are:
x-, y-, z-coordinates, yaw, pitch, and roll for Palm, Wrist and each joint. The distance from the palm to each finger is also added. This gives a vector size of 152 for a single frame.

The gestures should be able to be recognized all the time, so it was decided to evaluate a gesture every second, but use the data from the last three seconds. Each second should record ten full feature vectors, following equation 1, then a feature vector containing 4560 features will be sent to the API ever second. The handling of the feature vector in the API is described in 4.4.

$$f_{total} = f_{frame} * times_{prSec} * t \Rightarrow f_{total} = 152 * 10\frac{1}{s} * 3s = 4560 \quad (1)$$

The API should then respond with a classification and a certainty for the classification. The game should then respond accordingly to this classification and certainty. It is possible to save the feature vector in a CSV file instead of sending it to the API. This is useful when collecting data that should be used for training the machine learning algorithms.

4.4 API

The API was created as it was not possible to continuously send and receive messages from a python program started by a process. To bridge this shortcoming it was decided to create an API since it was not an option to start a new process for each invoke, since just starting the process and loading a single saved model took more than a second. The API contained all the models with all of them loaded so the machine learning algorithms could be changed run-time from the client. The API is created so it takes a feature vector and a machine learning algorithm, and then it responds with a classification and the certainty of that classification.

4.5 MACHINE LEARNING ALGORITHMS

The following subsections will describe the different machine learning algorithms implemented.

4.5.1 *Nearest Class centroid*

NCC is a classification scheme, which trains by calculating the centroid for each class. This centroid is the mean vector of each class. The Euclidean distance from a sample to all centroids are calculated and used to classify each test sample. The test sample is assigned to the class with the smallest distance.

4.5.2 *Nearest Subclass Centroid*

The NSCC is an extension of the nearest class centroid, where each class is allowed multiple subclasses. This will normally increase the accuracy but at the cost of more computations. The splitting of the classes into subclasses is by using k-means which is an unsupervised clustering method.

4.5.3 *Support Vector Machine*

SVM inserts a hyperplane to separate the classes. The hyperplane is of $p-1$ dimension where p is the dimension of the data. This means that for a feature vector of two dimensions the hyperplane will be a one-dimension hence a line. For a three-dimensional feature vector the hyperplane will be a two-dimensional plane and so on. This is illustrated in Figure 4.

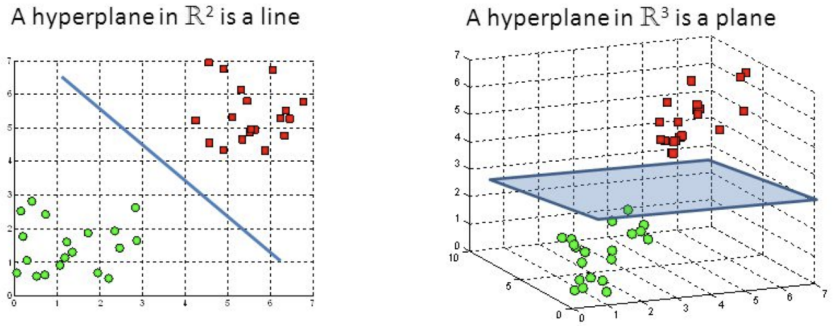

Figure 4: Hyperplane in two and three dimensional space[7].

The hyperplane is placed with the greatest distance between both classes, this is calculated by the Maximal Margin Classifier. The hyperplane will have a margin on both sides where to the data inside this margin will contribute to its location. The size of the margin is determined by the hyperparameter C[7], which describes the error tolerance allowed. The error is calculated as the distance of the data to the hyperplane. SVM can be done for either One-Versus-One (OVO) or One-Versus-Rest (OVR), where the first creates a binary classifier for each class pair and the second creates one binary classifier for each class versus all other classes.

4.5.4 *k-Nearest Neighbour*

k-NN is a straight forward approach where the samples are classified depending on the classes of its nearest neighbours. K is the number of neighbours taken into consideration when performing the classification. When the neighbour set consists of different classes the classi-

fication can be performed multiple ways. One where the classification is based on the majority of the neighbour classes. Another approach is to weight the neighbours' votes by the distance from the sample to the neighbours. This will make the neighbours closest to the sample have a greater influence on the classification of the sample. Equation 2 shows an example of weighted distance voting [4].

$$Vote(y_j) = \sum_{c=1}^{k} \frac{1}{d(q,x_c)^n} 1(y_j, y_c) \tag{2}$$

- Let y_j be the j labeled class of Y.
- Let y_c be the label of the c neighbour.
- Let q be the unclassified sample.
- Let x_c be the c neighbour.

Thus the vote for class y_j by neighbour x_c is 1 divided by the distance to that neighbour, i.e. $1(y_j, y_c)$ returns 1 if the class labels match and 0 otherwise[4].

4.5.5 *Backpropagation Perceptron*

BPP is a single layer Neural Network (NN). As in other NN, this method also has a weight attached to each neuron. During the training phase, the weights are updated to minimize the number of wrongly classified samples. Usually, in a binary classifier, the classes are labelled as $l_i \in 1$ and -1. Assuming the function in equation 3 has an optimal weight vector, thus classifying every sample correctly will mean that $f(\mathbf{w}^*, x_i) \geqslant 0$.

$$f(\mathbf{w}^*, x_i) = l_i \mathbf{w}^{*T} x_i \tag{3}$$

From equation 3 the misclassified samples will result in a negative value. These samples can then be used to express the overall error vector (χ) the error is then calculated in equation 4.

$$\mathcal{J}_p(w) = \sum_{x_i \epsilon \chi} -l_i w^T x_i \tag{4}$$

Optimizing $\mathcal{J}_p$ is done by calculating the derivative of it with respect to w. Which leads to the weights to be calculated by equation 5:

$$w(t+1) = w(t) - \eta(t) \nabla \mathcal{J}_p = w(t) - \eta(t) \sum_{x_i \epsilon \chi} l_i x_i \tag{5}$$

4.5.6 *Mean-Square-Error Perceptron*

Mean-Square-Error Perceptron is a single layer NN. Opposite to the Backpropagation the Mean-Square-Error (MSE) algorithm does not optimize to reduce the number of wrongly classified samples. The BPP might not always converge, consider the simple case of XOR where the samples cannot be linearly separated. Instead, it minimizes the error of the actual output to the wanted output of the network. When training, this method corrects the weight on each training sample in regards to minimizing the MSE. $\mathbf{w}$ is optimized by mapping training vectors $X = [x_i, ..., x_N]$ to arbitrary target values $\mathbf{b} = [b_1, ..., b_N]^T$, and finding the $\mathbf{w}$ which satisfies 6.

$$\mathbf{X}^T\mathbf{w} = \mathbf{b} \tag{6}$$

Minimizing the MSE corresponds to the optimization problem in equation 7[10, p. 47-48].

$$\mathcal{J}_s = \sum_{i=1}^{N} (\mathbf{w}^T\mathbf{x}_i - b_i)^2 = ||\mathbf{X}^T\mathbf{w} - \mathbf{b}||_2^2 \tag{7}$$

To find the most optimal $\mathbf{w}$, $\mathcal{J}_s$ has to be as small as possible. therefore to find the minimum of $\mathcal{J}_s$ the gradient of $\mathcal{J}_s$ is found and set equal to zero as shown in equation 8 and 9.

$$\nabla\mathcal{J}_s = 2\mathbf{X}\mathbf{X}^T\mathbf{w} - 2\mathbf{X}\mathbf{b} \tag{8}$$

$$\nabla\mathcal{J}_s = 0 \Rightarrow \mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{b} \tag{9}$$

From equation 9 $\mathbf{w}$ can be isolated as shown in equation 10.

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{b} \tag{10}$$

4.5.7 *Deep Neural Network*

The last method used is DNN, which is a fully connected neural network. DNN can be used in many different ways regarding which activation functions and loss function. The network has been constructed with inspiration from Fei-Fei Li's lecture about neural networks[11]. The network is constructed by five layers each containing a number of neurons in each layer. Using the ReLu and softmax activation function and the sparse categorical cross-entropy loss function.

4.6 AUGMENTED REALITY

The following section details the methodologies and design considerations of the AR.

4.6.1 *Projectile trajectory*

Central to the user's experience is the conceptualization of actually being able to shoot in a desired direction. The problem is twofold, as it requires a way of determining when a user wants to shoot as well as calculating which direction the user intends to shoot in. Determining when the user wants to shoot can be done in two ways, either by recognizing a certain gesture or by imposing a set of rules related to the hand(s). Alternatively, a mixture of both rules and gesture recognition can be used. The rules used can be implemented in many ways, such as

- Automatically, X seconds after conjuration
- When distance/angle between thumb and index finger is below a certain threshold (resembling a *finger gun*)
- When the velocity/acceleration surpasses a certain threshold
- When the velocity/acceleration surpasses a certain threshold and hand is brought to a stop (thus triggering the shot at the end of a thrust motion)

Determining the resulting trajectory could thereafter be done in several ways. In this project the pose of the hand and most recent hand positions were investigated as possible options for determining the best trajectory, inspired by the solution of Feuchtner [6]. The selected approach needed to be coupled with a fitting trigger in order to create a sensible method of shooting. The final result was using a gesture recognition trigger coupled with a hand pose trajectory calculation. Given more time, a better solution would probably be achievable by omitting the gesture recognition and instead use a mixture of recent positions and velocity changes for both triggering and calculating the trajectory. While this approach would require a high degree of parameter tweaking (which duration should be accounted for in the trajectory, should the previous hand positions be weighted according to timestamp/acceleration, etc.) it would allow the user to really throw an object and see it travel in a foreseeable direction. The resulting velocity of the projectile could additionally be scaled based on the speed at which the user threw the spell.

4.6.2 *Occlusion*

To enhance the AR experience occlusion made by the user's hands were introduced. The hands were chosen as they already came with a game object from Leap Motion on which a shader could be applied. Unfortunately, the hand models from the Leap Motion do not precisely overlay the real hands as seen from the camera, producing some results where the augmented objects either gets occluded slightly before or after the boundaries of the hands. This error was not corrected as this could be fixed by using an out of the box HMD. Another solution could be to include a shader where the shader boundaries would extend slightly past the hand models and use skin texture samples to determine whether or not an object was behind the hands.

4.6.3 *World Anchoring*

To create the illusion that a spell was thrown the spell needed to implement the properties of a real projectile. When an actual ball is thrown through a room, the ball gets detached from the person throwing it the instant it leaves the person's hand. The same property needed to be imposed on the spells so that any actions afterwards wouldn't affect it. Without any alterations the spell is translated by the rotation and movement of the user, making the spell look like an overlay rather than an augmented object. To detach the spell projectiles from the user, two approaches were taken into consideration:

- Room feature detection
- World anchoring

Room feature detection would be the hardest solution to implement. It would require a feature point extraction (SURF, SIFT, ORB e.g.) of each frame and use the translation and rotation of distinct features to correct the position of the projectile accordingly. While this approach would give the highest amount of freedom in terms of how and where the app could be used, it would also be the computationally heaviest.

World anchoring was chosen as it offered the best performance to complexity ratio of the possible solutions. This approach detaches the projectile from the user upon being cast and attaches it to an anchored point in the room. As the solution already included a target (a zombie on a Vuforia image target) this approach didn't introduce any complexities. World anchoring does, however, limit the use of the application as the projectile is only able to move relative to its anchor. World anchoring was chosen as it produced very good results whenever the anchor was present.

4.6.4 *Scaling*

In order to make the overall experience believable, the scaling of the augmented objects had to be taken into account. Initially, a default approach was implemented, where one world unit was used as one meter. This resulted in some oddities where the size of either spells or targets felt off, or the distance between objects seemed unreasonable. The scaling problem was avoided by scaling everything in relation to the user's hands. This process can be seen in figure 5 and pay notice to the similar size of the augmented hands and the hands of the zombie.

Figure 5: Scaling the zombie in reference to the virtual hands.

4.7 DESIGN CONCLUSION

The design ties the implemented classifiers together with a design for conjuring and casting spells. The game design describes functionality such as world anchoring, trajectory calculations, matching occlusion between actual hand and leap motion hand which together with the machine learning should give the user an AR gaming experience using machine learning. The machine learning models can be changed run-time which allows the user to experience how each model performs in terms of classification accuracy. The design thus supports hypothesis 1, *Augmented reality can effectively be used as a tool for comparing machine learning algorithms.* A comparison of the implementations of the different models will be performed in chapter 6 which support the hypothesis 2, *Neural Network is the best performing machine learning algorithm for hand gesture recognition in a game using leap motion.*

5 IMPLEMENTATION

5.1 OVERVIEW

In this chapter, the implementation of the API will be explained. Furthermore, the implementation of the different machine learning algorithms will also be explained. Afterwards the spells and trajectory calculations are discussed as well as the HMD. At the end of the chapter the AR is described.

5.2 API

The API is implemented in Python and uses the microframework flask[1]. The API only contains a single action method called features where it is possible to POST a feature vector and receive a classification and the certainty. The request should contain a query deciding what machine learning algorithm to be used. There are seven different algorithms to choose from. A more detailed description of the different algorithms can be found in section 4.5. When the API is started all 7 algorithms are loaded, from saved models so each of the algorithms can be called from a client with no delay.

5.2.1 *Machine learning*

Seven different machine learning algorithms have been trained for this project. Six of them using sklearn methods and the deep neural network using TensorFlow 2.0. Each of the algorithms has been implemented in the Classifiers.py which can be found in the appendix. They all contain four different methods; fit, predict, save and load. Fit is for training where a training set and the associated labels are taken as input parameters. Predict is taking an array of samples which should be classified. Save and load are used for respectively saving the model after training and loading the model when the API starts. The Support Vector Machine can be observed in listing 1. This is one of the shorter implementations and proves how sklearn enables simple implementations of machine learning algorithms.

1 http://flask.pocoo.org/

Listing 1: Support Vector Machine Implementation.

```
class SVM:
    def __init__(self):
        self.clf = svm.SVC(gamma='scale', decision_function_shape
            ='ovo', probability=True)

    def fit(self, train_data, train_lbls):
        self.clf.fit(train_data, train_lbls)
        return self

    def predict(self, test_data):
        probabilityData = self.clf.predict_proba(test_data)
        percentage = np.max(probabilityData, axis=1)
        classification = np.argmax(probabilityData, axis=1)
        return classification, percentage

    def save(self, version):
        pickle.dump(self, open(f"models/svmv{version}.sav", 'wb')
            )

    def load(self, version):
        self = pickle.load(open(f"models/svmv{version}.sav", 'rb'
            ))
        return self
```

Initially, the algorithms should just return a classification. After this worked well it was decided to extend the functionality to also include a certainty for the classification. This was not possible with the implementation chosen for the two Nearest Centroid methods, but since they were in the lower end of the accuracy it was decided that they should return a dummy percentage. The five remaining methods were extended so they returned a classification and the certainty. These results were then used in the game to decide what actions should be taken, if the observed certainty was below 70% the classification was discarded. The test results for the different machine learning algorithms is presented in Chapter 6.

5.3 GAME

5.3.1 *Interactive spells*

To amplify the feeling that a conjured spell originates from a users gestures it was decided that a spell should feel interactable in the gap between conjuration and dispatch. To achieve this effect a conjured spell has been implemented as a cluster of small elements, each gravitating towards the center of the hand used to conjure the spell. Each of the spell elements implements a custom script for the gravitational

effect as well as a rigidbody[2] to allow for internal collisions, making the spell as a whole act more like a unit. To further enhance the experience the hand models provided through Leap Motion were altered to be able to interact with the rigidbodies of the spell. This was done by implementing the Leap InteractionManager[3].

To stylize the spells to best resemble the different elements (fire, ice and organic) NVIDIA FleX[4] was investigated as an option. FleX allows for particle systems to resemble interactable matter such as liquids or volumetric smoke by using an NVIDIA GPU for real-time rendering. While this approach produced convincing visuals it was incompatible with Leap's InteractionManager, as neither Leap or FleX actually implements rigidbodies for the other to interact with. Abandoning this approach, each of the gravitating elements was instead stylized separately using Unity's built-in particle system.

5.3.2 *Trajectory calculations*

The final implementation of the trajectory used for firing a spell uses the normal vector from the palm of the hand with an adjustable offset. One of the earlier implementations used the previous positions of the hand, allowing for a projectile to be fired with a thrust motion. While the solution produced accurate results, it was abandoned as discussed in 4.6.1. The implementation of the latter approach can be seen in listing 2. It shows the continuous sampling of hand positions and how this was used to fire a projectile in the direction of a thrust.

2 https://docs.unity3d.com/ScriptReference/Rigidbody.html
3 https://api.leapmotion.com/documentation/v2/unity/unity/Unity.IE.InteractionManager.html
4 https://developer.nvidia.com/flex

Listing 2: Trajectory calculations using previous hand positions.

```
// Queue for storing 10 last hand position samples
private Queue<Vector3> previousPositions;
...
// Coroutine sampling hand positions from the past 500 ms
IEnumerator SavePreviousHandPos()
{
    while (true)
    {
        while (targetHand != null)
        {
            previousPositions.Enqueue(targetHand.transform.
                position);
            if (previousPositions.Count > 10)
            {
                previousPositions.Dequeue();
            }
            yield return new WaitForSeconds(0.05f);
        }
        yield return new WaitForSeconds(0.1f);
    }
}
...
// On shoot trigger
spellPrefab.GetComponent<BallSpell>().Shoot((targetHand.
    transform.position - previousPositions.Peek()));
...
// Shoot method on the projectile prefab
public void Shoot(Vector3 direction)
{
    GetComponent<Rigidbody>().AddForce(direction.normalized *
         customVelocity);
}
```

5.4 HEAD MOUNTED DISPLAY

The final result of the HMD can be seen in figure 6 here a side view is provided. It can be seen that the RGB-camera and Leap Motion is duct taped to a cap on top of each other.

This setup is not an optimal experience for the user because the user must use an external display to view the AR effects. But it is a good setup for proving the hypotheses.

Figure 7 illustrates the HMD seen from a front perspective. Here it can be seen that the Leap Motion is on top and the RGB-camera is on the bottom. This is tested to be the easiest and most stable way to stack the cameras on the HMD. It was important that the Leap Motion and the RGB-camera is pointing the same direction to make it less complicated, and that the offset was only on one axis.

Figure 6: Side view of the HMD.

Figure 7: Front view of the HMD.

Figure 8: Spell particles of fireball.

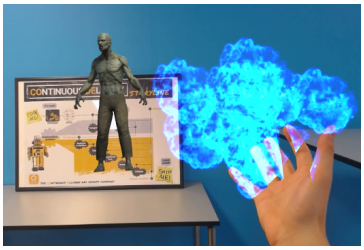

Figure 9: Spell particles of frost ball.

The Leap Motion is used for hand tracing and the RGB-camera is used for showing the real world that the spells can augment.

5.5 AUGMENTED REALITY

The AR consist of creating spell particles from three different spells as well as shooting it as a projectile. The particles gravitate towards the palm of the hand of the user to make the effect of the user have control and power like a wizard. There are three spells available, a fireball, a frost ball and a leaf ball the spell particles can be seen in figure 8, 9 and 10.

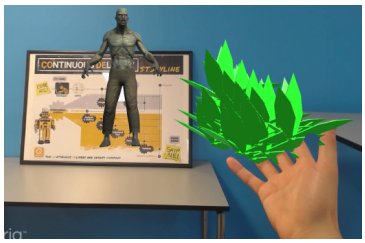

Figure 10: Spell particles of leaf ball.

The AR uses Vuforia to track image targets and show a zombie in front of the image target, the image target used can be seen on figure 11[5]. This makes the experience game like because there is something to shoot and kill. The zombie has animations when it stands still and

5 This image target is borrowed from Orbit lab

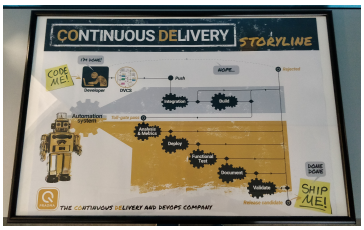

Figure 11: Image target used for spawning the zombie.

Figure 12: Image target with the spawned zombie.

he falls when it is hit by a spell to make it appear more alive. The zombie then resurrects three seconds after being hit for the player to shoot it again. The projectile explodes when the zombie is hit to give a satisfying feeling then a hit is made. If the zombie is not hit the projectile just keeps flying and after three seconds it loses its power and disappears. The zombie in front of the image target can be seen in figure 12.

To make the program run smother an automation of deleting the spell particles were implemented. If this was not implemented the program would run slower each time a spell was made because of the increase of memory usage. This automated deleting of spells waits for five seconds before deleting the spell from the object hierarchy.

6 EVALUATION

This section will first explain how the different models were optimized, then show the final comparison and then, in the end, it will be explained how the user tests were performed and the most important takeaways.

6.1 HYPERPARAMETER OPTIMIZATION

The implementation created for NCC and NSCC did not support confidence and the initial results were not on par with the other models. It was, therefore, decided not to focus on these two models in the experiments and optimizations.

All the training and test was done on the same 70% and 30% of the dataset for all tests. This was done by setting the random_state when doing the split, hereby the results should be comparable.

The hyperparameter optimization for SVM can be observed in table 1. For the stationary parameters, the following parameters have been used: Decision function=OVO, kernel=rbf, C=1, and all tests are running with "scale" as gamma parameter. From the table it can be concluded that by using a linear kernel or Radial Basis Function (RBF) with a C of 10 or more a result of 100% can be achieved. This is a satisfying result and one of these parameter settings should be used for the final test.

Table 1: SVM hyperparameter optimization.

Decision function						
OVR	OVO					
98.8%	98.8%					
Kernels						
linear	poly	RBF	sigmoid			
100%	99.7%	98.8%	64.7%			
C Parameter						
0.001	0.01	0.1	1	10	100	1000
66.6%	73.1%	86.5%	99.0%	100%	100%	100%

The default parameters for k-NN are neighbours=5, algorithm=auto, and weight=weight. From table 2 it can be concluded that the hyperparameter changes to the classification rate is minimal with the difference between the worst and the best is less than 1 percentage point.

Since higher neighbours give a better variability for certainty of the classification the six neighbors hyperparameter is chosen.

Table 2: k-NN hyperparameter optimization.

Neighbors				
1	2	3	4	5
97.4%	97.4%	96.9%	97.4%	97.2%
6	7	8	9	10
97.4%	96.9%	96.9%	96.2%	96.5%
Algorithm				
auto	brute	kd_tree	ball_tree	
97.2%	97.2%	97.2%	97.2%	
Weight				
uniform	distance			
97.2%	97.2%			

The hyperparameter optimization for BPP can be observed in table 3. Here parameter optimization has been evaluated using a stable learning rate in the first part and then with a decaying learning rate in the second part. It was found that as long a stable learning rate was used or the decay was very low, then a 100% correct classification was accomplished. Therefore the test was done with requiring a confidence of minimum 70%. From these results the stable learning rate of 0.5 was chosen for the final comparison.

Table 3: BPP hyperparameter optimization.

Learning Rate						
0.1	0.25	0.5	1	5	10	20
100%	100%	100%	100%	100%	100%	100%
Learning Rate with min confidence of 70%						
0.1	0.25	0.5	1	5	10	20
51.9%	56.2%	56.2%	56.2%	55.5%	55.5%	55.5%
Decaying Learning rate						
0.001	0.0025	0.005	0.01	0.05	0.1	0.5
100%	100%	99.8%	99.3%	88.8%	78.0%	28.8%
Decaying Learning rate with min confidence of 70%						
0.001	0.0025	0.005	0.01	0.05	0.1	0.5
39.2%	41.5%	26.2%	26.4%	5.4%	3.8%	0%

The hyperparameter optimization for MSEP can be observed in table 4. Here parameter optimization was done on the epsilon parameter

which is the learning rate. It was found that for all the tested values of epsilon a classification rate of 100% was found. It was, therefore, decided to test with a minimum confidence of 70% and 90% so a conclusion on what parameters to use could be found. It was found that an epsilon of 10000 would give the best results with both confidences.

Table 4: MSEP hyperparameter optimization.

Epsilon with min confidence of 70%				
0.0001	0.001	0.01	0.1	1
83.7%	95.9%	96.4%	96.5%	96.7%
10	100	1000	10000	100000
97.5%	98.8%	99.2%	99.7%	99.7%
Epsilon with min confidence of 90%				
0.0001	0.001	0.01	0.1	1
51,2%	66.72%	68.5%	68.7%	69.9%
10	100	1000	10000	100000
73.8%	80.0%	82.7%	82.7%	75.1%

For the DNN the hyperparameter optimization can be observed in table 5. Here three parameters where optimized: dropout percentage, size of layers, and learning rate. The default settings used for the optimization were dropout=0.1, learning rate=0.00001, and size=4560. The model used contained five relu layers, and one softmax and the loss function was a "sparse categorical cross-entropy" function. It was from the hyperparameter optimization found that with a size of 570 it was able to accomplice 100% on classification with confidence 70% and 90%. The 70% and 90% are not include in table 5. This parameter setting will, therefore, be used in the final comparison.

Table 5: DNN hyperparameter optimization.

Dropout					
0.01	0.025	0.05	0.1	0.25	0.5
99.8%	98.5%	99.8%	100%	100%	99.5%
Size of layers					
4560	2280	1140	570	285	
100%	99.0%	100%	100%	99.8	
Learning rate					
0.00001	0.0001	0.001	0.01		
100%	91.3%	69.4%	28.8%		

6.1.1 *Hyperparameter summary*

It was found that since only four classes were created for testing the models then performed really well. They actually worked so well, that in multiple cases the models could classify 100% of the test set correct, so it was hard to define what parameters should be used. This could be improved by adding more gesture classes and gestures which were more similar. This would increase the difficulty of classifying the gestures.

6.2 FINAL COMPARISON

Using the parameters found for each model the final comparison can be seen in table 6. The parameters used were:

- SVM with decision function OVO, with an RBF kernel and C-parameter of 100.
- k-NN with six Neighbours, auto algorithm and weights using "distance".
- BPP with learning rate of 0.5.
- MSEP with an epsilon of 10000.
- DNN with layer size of 570, dropout of 0.1 and learning rate of 0.0001.
- NCC with no optimized parameters.
- NSCC with three subclasses.

Since the confidence was used as a measurement for how good the gesture was, then this parameter should also be tested so a deliberate choice for this value could be taken. Therefore all the methods were tested with different confidence values. These tests can be seen in the lower part of table 6.

Table 6: Final comparison.

SVM	k-NN	BPP	MSEP	DNN	NCC	NSCC
Evaluation						
100%	97.4	100%	100%	100%	71.3%	87.5%
Evaluation with min confidence of 70%						
99.8%	92.3	56.2%	99.7%	100%	n/a	n/a
Evaluation with min confidence of 80%						
99.8%	90.3	20.8%	96.4%	100%	n/a	n/a
Evaluation with min confidence of 90%						
99.8%	83.7	2.8%	82.7%	100%	n/a	n/a
Evaluation with min confidence of 95%						
99.7%	83.7	0.5%	54.7%	100%	n/a	n/a

Following the results obtained in the final comparison, it was found that if multiple of the methods are really good at classifying the gestures. When the confidence is added at a parameter the classification

percentage starts dropping with the exception of SVM and DNN. They both keep really good results even when they should be more than 95% certain of a classification. Therefore, they would be the best candidates for gesture classification where the confidence is important and if a single method should be picked it would be the DNN.

The confusion matrix for the classifiers achieving 100% can be observed in table 7, the test classes distribution can here be observed. More misclassifications would have given a more interesting confusion matrix.

Table 7: Confusion matrix for 100% classification.

161	0	0	0
0	129	0	0
0	0	175	0
0	0	0	142

6.3 USER TESTS

There have been conducted user tests to evaluate how the models react to people whose hand data has not previously been seen by the model. The test setup consists of the user standing in front of the image target used with the HMD mounted. An external screen is put beside the user to show the augmented effects. The test consist of the user summoning the three spells twice and shooting the zombie. This way the machine learning models are tested over a number of spells and over a period of time. The user tests were done on six persons with satisfying results and some ideas to improve the user experience was found. A picture of a user test in action can be seen in figure 13

The results of the user tests showed that the threshold of acceptable confidence is to low, and can easily be tweaked to work better. The users sometimes created the wrong spell while transitioning between hand gestures. Another observation during the the tests were that the messages between the unity program and the API queue piled up over time. This means that the user can feel a delay between the hand gesture done and the program recognizing the gesture. This was a problem doing the last spells created by the users and was commented by a majority of the users. The overall conclusion of the users was that it worked well, but the AR was difficult to follow because of the augmented effects being on an external screen. But the effects and a shooting target was extremely fun and they could see the potential in the idea of the game. They also got an understanding of how the different algorithms performed, compared and how confidence adjustments made a huge difference when comparing the different

Figure 13: User test where the participant tries to make a fireball.

algorithms. However, the shooting direction was a bit difficult for the users at first.

The conclusion of the user test is that the threshold for confidence should be increased to a higher level, to lower the number of false positives. Furthermore, the lag when sending requests, and queuing up, should be improved by canceling the remaining requests when the gesture is approved. The shooting direction could be improved by tweak the parameters a bit making the shooting more intuitive for users. It was through the test observed that the users would get an understanding of how the different algorithms would compare and that some algorithms was better at correctly classifying the gestures.

7 CONCLUSION

Though this paper an AR game has been implemented to test the power of seven ML algorithms as gesture recognizer by using the Leap Motion data. The outcome is a product that allows a user to conjure three spells and shoot the spells through hand gestures. The product uses Vuforia to detect image targets to spawn a zombie that the user can shoot. The zombie is animated and reacts to the spells that is thrown at it. When the user make one of the hand gestures, particles will orbit the user's hand and the spell is ready to be thrown.

The NCC and NSCC has not implement confidence level because the initial results were not on par with the other models, therefore, those algorithms were quickly discarded as good candidates. The k-NN was the only algorithm that implemented confidence level that did not hit the 100% correct classification. The rest of the ML algorithms gave a 100% correct classification and by testing further showed that the DNN had the best confidence level overall, by ending at a 100% classification for the test set even with a confidence level of 95%. By this the hypotheses 2 can be confirmed in the case of AR-Wizard when the confidence is part of the evaluation.

The HMD used in this project was a Leap Motion and an RGB-camera mounted on a cap. This approach showed some problems in relation to viewing angle and offset of the two cameras. The optimal solution would be to use an out of the box HMD, this would prevent the viewing angle and offset problems. The out of the box HMD would also give a more immersive experience with the screen located in front of the user, instead of the use of an external screen that the user has to look at to see the augmented objects.

AR-Wizard prove that ML is a powerful tool in combination with AR to create an immersive experience for a user. The user tests showed that users are amazed by the AR-Wizard response on their hand gestures. The tests also showed that the users were interested in the final release date of the product and gave some improvement ideas. The interest and feedback from the users show that AR can effectively be used as a tool for comparing ML algorithms. This proves that hypotheses 1 can be confirmed. This was further confirmed by the users in the user tests using the ability to adjust on the machine learning algorithms and the confidence interval.

ACKNOWLEDGMENTS

A big thanks to Richard Drew the inventor of Adhesive tape the predecessor to Duct Tape, without him we would not have been able to create the HMD. A big thanks to Orbit lab in Aarhus University for letting us borrow an extra Leap Motion, an image target, and the usage of space to make experiments.

BIBLIOGRAPHY

[1] Safa Ameur, Anouar Ben Khalifa, and Med Bouhlel. "A Comprehensive Leap Motion Database for Hand Gesture Recognition." In: Dec. 2016. DOI: 10.1109/SETIT.2016.7939924.

[2] Teak-Wei Chong and Boon Giin Lee. "American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach." In: *Sensors* 18 (Oct. 2018), p. 3554. DOI: 10.3390/s18103554.

[3] Alex Colgan. *How Does the Leap Motion Controller Work?* Aug. 2014. URL: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/.

[4] Padraig Cunningham and Sarah Delany. "k-Nearest neighbour classifiers." In: *Mult Classif Syst* (Apr. 2007).

[5] Youchen Du, Shenglan Liu, Lin Feng, Menghui Chen, and Jie Wu. "Hand Gesture Recognition with Leap Motion." In: *CoRR* abs/1711.04293 (2017). arXiv: 1711.04293. URL: http://arxiv.org/abs/1711.04293.

[6] Tiare Feuchtner and Jörg Müller. "Extending the Body for Interaction with Reality." In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: ACM, 2017, pp. 5145–5157. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025689. URL: http://doi.acm.org/10.1145/3025453.3025689.

[7] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. Accessed: 2019-05-18. June 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[8] Caroline Guardino, Ching-Hua Chuan, and Eric Regina. "American Sign Language Recognition Using Leap Motion Sensor." In: Dec. 2014. DOI: 10.1109/ICMLA.2014.110.

[9] Nate Hagbi, Raphael Grasset, Oriel Bergig, Mark Billinghurst, and Jihad El-Sana. "In-Place Sketching for Augmented Reality Games." In: *Comput. Entertain.* 12.3 (Feb. 2015), 3:1–3:18. ISSN: 1544-3574. DOI: 10.1145/2702109.2633419. URL: http://doi.acm.org/10.1145/2702109.2633419.

[10] A. Iosifidis. "Introduction to Machine Learning: Course notes." In: (Nov. 2017).

[11] Fei-Fei Li, Justin Johnson, and Serena Yeung. "Neural Networks and Backpropagation." In: (2019).

[12] Wei Lu, Zheng Tong, and Jinghui Chu. "Dynamic Hand Gesture Recognition With Leap Motion Controller." In: *IEEE Signal Processing Letters* 23 (Sept. 2016), pp. 1–1. DOI: 10.1109/LSP.2016.2590470.

[13] G. Marin, F. Dominio, and P. Zanuttigh. "Hand gesture recognition with leap motion and kinect devices." In: *2014 IEEE International Conference on Image Processing (ICIP)*. Oct. 2014, pp. 1565–1569. DOI: 10.1109/ICIP.2014.7025313.

[14] Chetna Naidu and Archana Ghotkar. "Hand Gesture Recognition Using Leap Motion Controller." In: *International Journal of Science and Research* 5.10 (Oct. 2016). ISSN: 2319-7064. URL: https://pdfs.semanticscholar.org/44d9/7bc5fe16862a88935de31e231c53737a35d3.pdf.

[15] Wei Zeng, Cong Wang, and Qinghui Wang. "Hand gesture recognition using Leap Motion via deterministic learning." In: *Multimedia Tools and Applications* 77.21 (Nov. 2018), pp. 28185–28206. ISSN: 1573-7721. DOI: 10.1007/s11042-018-5998-1. URL: https://doi.org/10.1007/s11042-018-5998-1.

[16] Bingqing Zhang and Wenfeng Hu. "Game special effect simulation based on particle system of Unity3D." In: *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. Wuhan, China: IEEE, 2017, pp. 595–598. ISBN: 978-1-5090-5507-4. DOI: 10.1109/ICIS.2017.7960062.