AR Puppeteering

Bernard Lupiac and Luiz Velho (supervisor)

Technical Report TR-19-02 Relatório Técnico

February - 2019 - Fevereiro

The contents of this report are the sole responsibility of the authors. O conteúdo do presente relatório é de única responsabilidade dos autores.
1 Abstract

This report describes the components of the AR Puppeteering project, how to use them and how they work. The goal of the project is to allow the manipulation of a marionette with both hands using the Leap Motion with augmented reality.

2 Motivation

The objective of this project is to develop an application that lets puppeteers make performances with nothing but their hands. It aims to preserve the expressiveness and know-how of classic puppeteering, but at the same time, to look at it in a different perspective with the help of augmented reality.

Because of this, there are a few points that are very important for this application.

- Its controls need to be reliable, so the user can trust it to always respond in the same way to the same inputs.
- They should also be as intuitive as possible, because the user should feel he’s directly manipulating the marionette, and not a program.
- The marionette should make realistic movements. Because of this, it needs to be stable.
- It should also be expressive and organic in its movements. To attain this, the range of possible movements should be varied, and the user should be able to control them with precision.
3 Physics Interaction

The first iteration of this project used a physics-based system. It is not used in recent versions, but the code exists in the repository in case it ever needs to be revisited. It draws a lot of ideas from this project: [Bot+14].

This system uses strings that connect the user’s hand and the limbs of the marionette. The strings’ movement is done with rope physics. In a first step, the strings connected each fingertip to one of the marionette’s limb. But after some experimentation, it was found that the movements were awkward and their range was very limited.

The next iteration of this system was closer to [Bot+14]. It attaches a crossbar to the user’s hand. Because of this, there are two types of movement possible, that can be done separately or combined:

- By moving the center of the palm, the marionette moves evenly.
- By tilting the hand, the limbs can move in many different manners.

This iteration was much better than the previous: movements were more reliable and less awkward. However, there were still some important problems. For instance, limbs couldn’t be moved individually, and the limited range of movements the human hand can make could still be felt. Because of those limitations, the next step was to try another system: one based around animations.

4 Animation Interaction

Because of Unity’s frameworks and tools for animation, this could be done with very little overhead. Animations made for humanoids could be imported and used directly. Asides from Unity’s tutorials, animations from this database were used in the project: [Car+00]. The manner in which Unity’s tools were used is explained in detail in Section 6.

In comparison with the previous physics system, this one has many advantages:

- Physics systems can be very unstable, especially when many bones and ropes are used. This problem is absent in an animation system, since there aren’t any constraints to be satisfied (except for the inverse kinematics part, which handles just a handful of bones).
- It allows the marionette to accurately perform complex movements like dancing and running, without too much effort. On the other hand, since animations are prerecorded movements, they can end up getting repetitive and limit the expressiveness of the marionette. One way to avoid this could be to blend many different, but similar animations and let the user control the weight of each one.
- With an animation system, it is also easier to circumvent the hand’s movement limitations.

5 Gesture Recognizer

5.1 Motivation

In this application, there are other interactions possible besides the direct manipulation of the marionette. Manipulations such as changing the type of control, resetting the application, or configuring the types of movement.

The application needs to let the user make all these interactions using only his hands. The solution that was chosen to make this possible was a gesture-based interaction system that uses a gesture recognizer.
To allow the system to be very flexible with the gesture it uses, and still be accurate, the recognizer was implemented using a machine learning technology called Support-Vector Machines (SVM).

5.2 Implementation

There is an impressive amount of papers on the subject of gesture recognizers for the Leap Motion. This implementation is based on [GFP14] and [You+17], because of the level of detail given for feature extraction and the settings used for the SVM respectively.

5.2.1 Features

From the papers mentioned above, the features used in this implementation for the classification are:

- The angle of each fingertip relative to the palm.
- Distance of each finger from the center of the palm.
- Distance of each fingertip from the plane created by the palm.
- Matrix of distances of all fingertips.

5.2.2 SVM Parameters

The SVM was implemented using Accord.NET [Sou+14], which is a C# library for machine learning. The settings of the machine are very similar to the ones described in section 5 of [You+17]. In short, it is a multi-class SVM, that uses grid-search to find hyper-parameters C and $\gamma$.

5.3 Learning

The different types of gestures the recognizer can learn aren’t hardcoded in any way. By redoing the learning routine, any gesture can be used in the application (given the limitations of the Leap Motion as a sensor).

The learning routine consists of capturing the hand in the desired positions many different times. Afterwards, all these examples will be used to determine the SVM. An in-depth guide of this routine can be found in the readme of the project.

After the machine is set up, it is serialized and saved to a file, that will be reused from that point on in the application. Since a file is generated, it is possible to make backups of previous SVMs and to easily switch between them.

5.4 Result

Once the machine is ready, it is very easy to use. It works as a function: its only inputs are the same features described in Section 4.2.1 extracted from a hand. The output is the prediction made by the machine: the ID of the gesture the machine has recognized, asides from some other data, like the reliability of the prediction.

6 Movement Modes

It is possible to move the marionette in three different manners. It is necessary to use gestures to navigate between them.
The application starts in a “neutral mode”, where the marionette doesn’t move. To enter in any of the movement modes mentioned below, it is necessary to make a specific gesture. To exit any of those modes, the user needs to make another specific gesture. Finally, if there is any need to reset the marionette or its position, there is also a reset gesture that is made with both hands.

The architecture of the application also allows for two of these movement modes to be made at the same time (one in each hand), even if it isn’t implemented in the current version.

6.1 Head Movement

The movement of the head is done using Unity’s inverse kinematics (IK) system. The marionette will look in the same direction as the direction of the palm detected by Leap Motion. For example, if the palm is facing up, the marionette’s head will be looking at the ceiling.

6.2 Arm Movement

It is very straightforward to control the marionette’s arms: it will move its arm in the exact same way as the user. The arms’ movement also uses Unity’s IK system, but in a more complex manner. Each arm has two target positions: the arm and the elbow. The IK system will then calculate the orientations of the marionette’s arms that are closer to the targets given. Adding the elbow’s position results in a much more realistic and smoother arm movement.
Figure 3: Arm movement in the application

To increase the accuracy of the movement, it is also necessary to calibrate the length of the arm and elbow according to the user’s.

6.3 Locomotion

The walking and running controls were subject to many iterations. The last iteration up to this point is stable and intuitive.

6.3.1 Input

It works in the same manner as a joystick in a video-game. But instead of receiving x and y coordinates from an analog stick, it receives the same information from the Leap Motion.

Upon entering the locomotion mode, a sphere will appear on the position of the hand that will control the marionette. It stays in the same position, but it can be moved using a gesture with the other hand.

Figure 4: Sphere that appears in the movement mode

The sphere acts as a dead-spot for the movement. Once the center of the palm is inside it, the received x and y inputs will be 0.

Figure 5: Hand inside the dead-spot in the locomotion mode

The sphere also acts as a reference, because its center is very important to determine the direction and intensity of the movement. They will be calculated using the vector that starts in the center of the sphere and...
goes in the direction of the center of the palm. The vector is then projected on the $xy$ plane. Its direction determines the direction of the movement, and its length determines the intensity of the movement.

![Figure 6: Sketch of how the movement is determined](image)

**6.3.2 Character controller**

As said above, the received input is practically the same from any joystick controller. Because of this, it was very easy to find tutorials on how to implement sophisticated methods of controlling the character. It also helped to find decent quality animations that could be used legally.

For the character controller, the following tutorial was used as a guide: [McE13]. This tutorial uses Unity’s Mecanim system, which is an animation tool and state machine. It also uses Unity’s blendtrees, that allow the interpolation of many different animations given a few parameters.

![Figure 7: State machine of the application’s movement manager in Mecanim](image)

Finally, the resulting character controller moves the character relative to the cam-

![Figure 8: Locomotion blendtree of the application](image)
era direction. With this, when the user’s hand moves to the left, the character always moves to the left of where the user is looking. The character’s position and orientation don’t have to be taken into account, unlike other movement schemes (tank controls, for example). It becomes much easier to manipulate the character because of this, and similar character controllers have been used in the majority of 3D 3\textsuperscript{rd} person games since the times of Mario 64 because of it.

Figure 9: Marionette running

7 Conclusion and Future Works

The application presented in this report fulfilled many of its initial objectives:

- It is possible for the marionette to make many different types of complex movements.
- The controls and gestures are relatively intuitive, and can let the user express an emotion through the marionette.
- The controls and the marionette are reliable and stable.

However, there are still some points that need further work:

- After feedback from a professional puppeteer, it was found that many different types of movement are still needed to make a good performance.
- It is still necessary to implement a way for the public to visualize the puppet, either through augmented or virtual reality, with devices like headsets for example. This entails incorporating a network component in the application.
- Another interesting enhancement would be to make a puppet that is controlled by many different people. Either through many Leap Motions in the same computer, or through many computers with one Leap Motion each.
- The current gesture recognizer only detects static gestures. Detecting dynamic gestures, although probably much harder, would make the possible gestures much more intuitive.
- Before its release, a fine-tune pass is still needed in order to maximize the stability of the controls and the accuracy of the gesture recognizer.
- As a more low-level enhancement to increase the accuracy of the gesture recognizer, the raw images from the Leap Motion can also be used as a feature[You+17], at the cost of performance.

8 Acknowledgements

The data used in this project was obtained from mocap.cs.cmu.edu.

The database was created with funding from NSF EIA-0196217.

The marionette puppet was taken from https://sketchfab.com/models/
with the addition of eyeballs.

References


