# Stylus-Based Gesture Recognition and Model Interaction Interface on zSpace Virtual Reality System

Nishad Gothoskar[1] and Clifford Champion[2]

*Abstract*— **This paper documents a project pursued over 2 months as a Software Engineering Intern at zSpace Inc. The pursuit was of the enablement of the zSpace technology to both recognize and respond to gesture-based commands from a user, principally while operating a stylus. This project would be created using C#, Unity Engine, Visual Studio, and the existing zSpace API. This would be done by the development of an API designated *zSpace.Gestures* that would handle all elements of data collection, preprocessing, recognition, and output. Following this, the user could simply appropriate a desired result for the recognized gesture.**

## I. INTRODUCTION

Virtual Reality is meant to be both natural and immersive: an extension of our current reality. Therefore, it seems befitting that a user's interaction with a Virtual Reality Interface is both natural and immersive. The existing technology of the PC is (in a Human-Computer Interaction perspective) archaic. The keyboard and mouse have been used since the early 60's at the inception of the Personal Computer, and while the PC developed at incredible speed with respect to efficiency and processing, interaction through keyboard and mouse has not evolved at all. This has bound us to Operating Systems and Interfaces that are centered around Discrete I/O's: Key Presses, Mouse Clicks, Button Presses. With Virtual Reality, we cannot restrict ourselves to such a limited method of interaction and IO. The zSpace System involves interaction through a stylus. The advantage of stylus interaction is the ability to move in 3D space and to give the natural pen-like feeling making it a perfect mode of input in a Virtual Environment. But further, in order to make the next generation of VR truly immersive, we want users to be able to interact with objects the way they do in real life: lift them, spin them, stretch them, group them, remove them, etc. In reality, interaction with objects is associated with actions and motions. The inspiration for our project was to bring the natural, innate interaction, through actions, motions, and *gestures*, into the field of Virtual Reality and Environments.

## II. GOALS AND RISK ASSESSMENT

Implementing a novel method of interaction in an interface where it was not previously present poses various risks. The risks associated with this project primarily involve ensuring that this extension of interaction adds to the usability and quality of the interface rather than takes away from it.

The project is highly dependent on an accurate recognition algorithm, however also, an algorithm that is sufficiently tolerant of user imprecision (inherent to gestures in free space). Without this, the project may have little or no effect. Therefore, the major risk in this project is this algorithm. Implementing it properly will require techniques of machine learning, pattern recognition, and ideas used in OCR. However, given that we will have hundreds or thousands of data points in 3 dimensions to represent a single gesture, we recognize that some components of the pattern data collection, preprocessing, and recognition will be complex. Therefore, we made it a priority to ensure that the implementation is efficient enough to minimize latency and response time as to not take away from the user experience.

Another issue involves the quality of our input data. With traditional OCR, algorithms are usually given typed or written characters. Given that the gestures will be given freehanded in 3 dimensional space, the quality will be much worse than even hand-written characters. We would like to populate the set of recognized gestures, however at the same time we must be able to recognize them accurately. This fine line between two pattern's differentiability is questionable and depends on the strength of the algorithm we can develop.

Understanding the major risks inherent to the project is important. We needed to keep them in mind as we took each step in our work, in order to ensure quality of our final project.

## III. APPROACH OVERVIEW

Our project essentially involved two components: recognition algorithm & interaction interface. The former required about 6 weeks while the latter required about 2. But each was equally important since an interface was useless without the algorithm behind it and an algorithm was useless without an application of it. The steps are outlined below:

### A. Recognition Algorithm

- Selecting the set of recognizable gestures
- Choosing proper/valuable data features to extract
- Applying proper preprocessing
- Creating the neural network architecture
- Training the neural network

### B. Interaction Interface

- Creating a state machine
- Designing the user interface

[1]N. Gothoskar is a Software Engineering Intern at zSpace Inc. and a student of Mathematics and Computer Science at Carnegie Mellon University `ngothosk at andrew.cmu.edu`

[2]C. Champion is a Software Engineering Manager at zSpace Inc. `cchampion at zspace.com`

*Note: In this paper, I will use **effect** to refer to what each gesture will do with respect to the interface (for example the effect of a C gesture is a copy)*

We understood from the start that this project would require methods of Machine Learning and some type of Learning Algorithm. The complexity of the problem made it such that a hard-coded solution would not only be complicated but also extremely unreliable.

## RECOGNITION ALGORITHM

## IV. SET OF GESTURES

The set of gestures was essential to the project. It was necessary to select gestures that were natural to users as well as gestures that could easily be associated with a corresponding effect.

However, we also had to ensure that the set of gestures was not "crowded". Meaning, we didn't want to have many gestures that would be hard for an algorithm to distinguish from one and other.

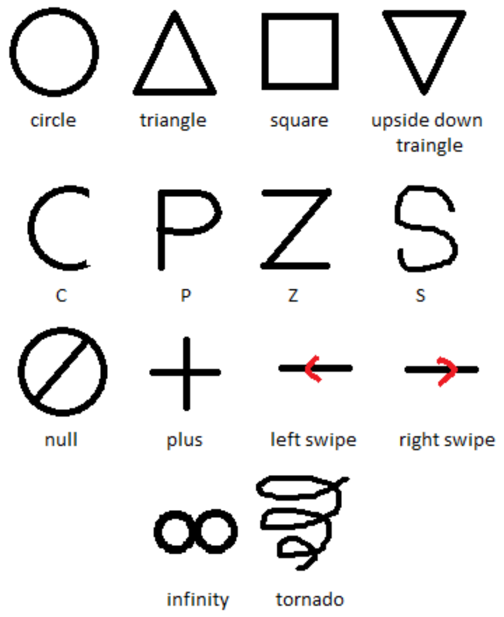


Fig. 1. The final set of gestures for the algorithm to learn and recognize

The set of gestures shown in Figure 1 seemed fitting to the set of *effects* which I wanted to implement and in addition, they seemed different enough to be recognized by an algorithm.

Further, it seemed that with these gestures, I could eliminate the need for any mouse or keyboard interaction. All interaction through my interface could be properly and fully controlled by only stylus-based interaction.

## V. FEATURE EXTRACTION

Those familiar with Artificial Neural Networks know a common structure involves an input layer, a hidden layer, and an output layer. But a neural network is useless if the input layer is not provided with significant information that can be synthesized by the network. Therefore, an essential component to our recognition algorithm was our data collection and choice of feature extraction.

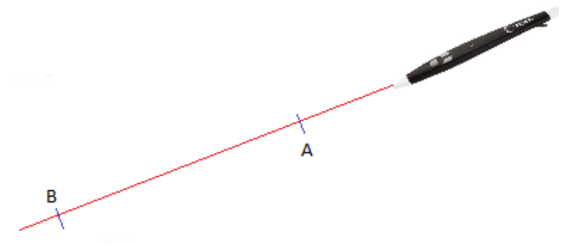


Fig. 2. Stylus Data Collection Points

As shown in Figure 2, after finding the pose of the stylus with its position and orientation, we can cast a ray with origin at the stylus tip and direction parallel to the stylus. For our data, we chose to collect data at two points, designated A and B in Figure 2, on the ray. Now, using this information we had to decide on a set of features to extract from this data. The features were chosen such that each feature had a specific role in differentiating gestures.

*A method which we found helpful in our project to assist with our choice of features, was to first attempt to hard code a decision tree style algorithm. This forced us to look for specific defining features of each gesture. This method allowed us to select relevant and effective features to extract. Eventually when we implemented learning, these hard-coded thresholds and weights were improved and perfected and our careful selection of features allowed for accurate recognition.*

Ultimately after graduating from a hard coded decision tree to a general neural network, what follows are the features we ultimately used.

The first and most important feature we extracted was the physical appearance of the completed gesture. We conveyed this by first resizing and fitting the points to a unit square and then converting each point to its corresponding pixel. (The method of doing this is described in Section VI.) We converted to pixel data at 4 different resolutions: 32x32, 16x16, 8x8, 4x4. We did this because through experimentation, we saw that in each instance the effectiveness of high resolutions and low resolutions vary. Therefore, we saw it worthwhile to include various resolutions.

Next, we used bounding box and convex hull algorithms to gather more in depth information about a specific gesture. We extracted the area and perimeter of the hull along with a ratio of those two. This ratio is significant because, for example, the ratio for a square is 1/4s (s is side length) while for a circle it is r/2 (r is radius). In addition, we included the ratio of the height to the width of the bounding box. This served to allow us to recognize flatter gestures like swipes from ones like circles and squares.

Finally, we had to include some data on the time dependence of the gesture. For example, the start and end positions of the gestures were essential. First we included the distance from the start point to the end point. Shapes like C, P, Z, S did not have the same start and end position, while Circle, Triangle, and Square did, so this data was useful. We also included the angle of the vector from the start point to the end point because this is differentiating feature in gestures like the Z and S.

In our project, we discovered that when we want to convey that a gesture went in the left direction or the right direction, it was necessary to divide that information into two features. One which took the value 1 if the gesture was left and 0 if right. And another feature which was the opposite(1 if right and 0 if left). This if because if we only expressed it as 1 and 0 a weight assigned to the 0 would be useless in a Neural Network that applies dot products (as that output would always be 0). Therefore it was also necessary to include its analogue.

## VI. PREPROCESSING

For our project, we took various preprocessing steps to prepare the data we had collected for feature extraction. Since we were collecting data points in 3 dimensions we needed to find a method to flatten it into 2 dimensions. A common method for doing this is PCA (Principal Component Analysis). However, this runs the risk of rotating the gesture which could cause issues for such shapes as the Triangle and Upside-Down Triangle. Therefore, we devised another approach for constructing a new basis with a rotation invariant.

First, we compute the mean and covariance of the points. This allows us to compute the Principal Component Axes. We then take the 3rd axis found from PCA. This is the axis of least variance (analagous to the normal vector to the plane on which the gesture is). Let us call this `zcomp`. But we want to make sure that this component points towards the user. So, we simply check the sign of the z component of this vector and negate it if it points into the screen, ensuring that its direction is out of the screen. Now the construction of the orthonormal basis is shown below.

$$ycomp = zcomp \times <1,0,0>$$

$$xcomp = ycomp \times zcomp$$

This forms our orthonormal basis composed of `xcomp`,`ycomp`, and `zcomp` as shown in Figure 3.

Now we create a matrix map using these vectors arranged column-wise. Using this map we can transform each point in our original space to our new basis and then ignore each z component. This gives us points in 2 dimensions that represent our gesture.

Now we need to normalize these points. Given that a gesture can be drawn with any size and it should still be recognized, we will normalize the points by calculating a bounding box. Since a user may want use the size of the gesture in designing an effect, we will remember and later

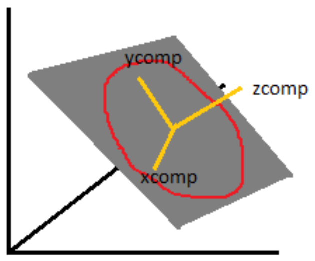


Fig. 3. Representation of basis formation

output the size of this bounding box as an output parameter. Then, we take the corners of the bounding box and map them to the corners of a unit square [0,1] x [0,1] creating an Affine Transform and normalizing the points.

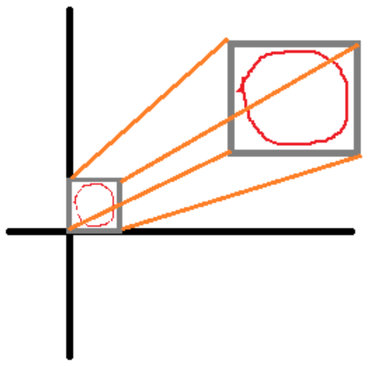


Fig. 4. Representation of the Affine Transform

Then we will use existing algorithms for finding Convex Hull to apply the steps we described in the previous section.

Finally, as described in the previous section we will convert each of the points to their corresponding pixel using the following method where `res` is the resolution at which we want to convert the points. As shown, the pixels don't only take on only values of 0 and 1. If two points correspond to the same pixel, then its value will become 2.

```
for (int i = 0; i < set.Length; i++)
{
  float x = set[i].X;
  float y = (1 - set[i].Y);
  int xa = (int)(x * res);
  int ya = (int)(y * res);
  if (xa == res)
  {
    xa = res - 1;
  }
  if (ya == res)
  {
    ya = res - 1;
  }
  features[res * ya + xa + 1] += 1.0;
}
```

## VII. Neural Network

The next step in the process was designing the recognition network. Our design was a Multi-Class Multilayer Perceptron (Artificial Neural Network).

Our first concern is to design a network such that its computation is very efficient. Our second concern is to ensure that the structure is easily serializable and deserializable in order to make it simple for users to load the weights. And our final concern is to make the computation done "behind the scenes" and provide the users with a simple interface to use the network's functions.

```
internal class NeuralNetwork
{
  public double LearnRate { get; set; }
  public List<Neuron> InputLayer { get; set;
      }
  public List<Neuron> HiddenLayer { get;
      set; }
  public List<Neuron> OutputLayer { get;
      set; }

  ...

  public void Train(params double[] inputs)
  {
    int i = 0;
    InputLayer.ForEach(a => a.Value =
        inputs[i++]);
    HiddenLayer.ForEach(a =>
        a.CalculateValue());
    OutputLayer.ForEach(a =>
        a.CalculateValue());
  }
}
public class Neuron
{
  public int id;
  public bool change;
  public List<Neuron> InputNeurons { get;
      set; }
  public double[] InputWeights { get; set; }
  public double Bias { get; set; }
  public double Gradient { get; set; }
  public double Value { get; set; }

  ...

  public virtual double CalculateValue()
  {
    double Sum = 0;
    for (int i = 0; i < InputNeurons.Count;
        i++)
    {
      Sum += InputNeurons[i].Value *
          InputWeights[i];
    }
    return Value =
        NeuralNetwork.SigmoidFunction(Sum +
        Bias);
  }
}
```

The NeuralNetwork class also contains a constructor that creates the NeuralNetwork from a ".txt" file from disk using a 3rd party JSON serializer. The structure is free of circularities or circular references.

We used this structure to design our network. We selected a hidden layer of 500 nodes and an output layer of 14 nodes, representing the 14 gestures. But through various tests we ran on our network we made some discoveries to significantly improve recognition, training precision, and accuracy. These discoveries are associated with the fact that a large amount ($32^2+16^2+8^2+4^2$) of the input layer nodes are associated with the pixel data, while there are only about 10 other features in the feature vector. These 10 features are "outweighed" by the large amount of other features and the discoveries about adjusting for this are discussed in the following two sections.

### A. *Layer Bypass*

A term in Neural Networks is "fully connected", meaning that every node in one layer is connected to every node in the previous layer. But for such a feature vector described above which is so large and weighted with one type of data, the data in the last few nodes may be obscured as it passes through the hidden layer. Therefore, it may be necessary to adjust this fully connected structure to better fit the problem at hand. I then created my network such that the last nodes bypass the hidden layer making the hidden layer not fully connected to the input layer, but ensured that the output layer was fully connected to the hidden layer.

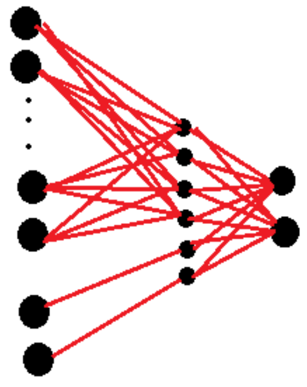


Fig. 5. Example of a Layer Bypass shown in the last 2 nodes of the input layer

### B. *Feature Reinforcement*

We observed that our recognition algorithm was having trouble recognizing left and right swipes. Given that the direction of the gesture was only given through 2 features of the feature vector, we assumed that the Layer Bypass was not enough to make it visible to the network. So in addition, we simply repeated those components of the feature vector 100 times each, having 100 nodes representing the same piece of data. While this seemed very repetitive, it was necessary in order to allow that data to be visible within a feature vector of greater that $32^2+16^2+8^2+4^2$ features.

## VIII. Training the Network

In order to train our network we had to collect sample data. However, the manner in which this data was collected was important. Since we wanted, in the end, for the gestures to be used extremely naturally and almost involuntarily, it was necessary to collect data in this same manner.

To do this, we created a simple music application that forced users to draw gestures in rapid succession, each gesture playing the next note in a familiar or popular song. This recreated an environment where a user didn't think much about drawing the gesture but rather just completed it as quickly as possible. The music served as a proper distraction for the user and therefore, we were sure that our data was representative of how a user might perform a gesture in a complete interaction interface.

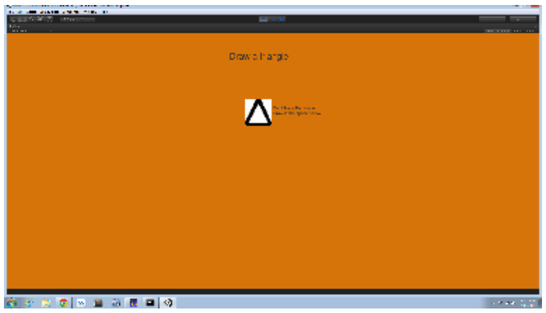


Fig. 6. Screenshot of our data collection application

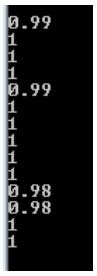
With this application we collected about 120 training examples of each of the 14 gestures. However, some of the training examples were not reliable in that users clearly did not draw the intended gesture. We created a method of visualizing each set of points and after removing this extraneous examples, set we were left with about 100 training examples of each gestures.

Then, using our existing Neural Network architecture, we used this method of back propagation.

```
public void BackPropagate(int g)
{
  ...
  OutputLayer.ForEach(a =>
      a.CalculateGradient(targets[i++]));
  HiddenLayer.ForEach(a =>
      a.CalculateGradient(OutputLayer));
  HiddenLayer.ForEach(a =>
      a.UpdateWeights(LearnRate));
  OutputLayer.ForEach(a =>
      a.UpdateWeights(LearnRate));
}
```

We ran our training at a learning rate of 0.02. And were able to converge to the following training set accuracy

## Interaction Interface



```
0.99
1
1
1
0.99
1
1
1
1
1
0.98
0.98
1
1
```

Fig. 7. Accuracy for 14 gestures after training

## IX. State Machine

The next major step is in designing the interaction interface on which the gesture recognition can be used. The state machine is important to the interface because it allows us to manage the necessary data properly. A gesture is initiated by a button press and ended by a button release. Therefore, our state will only be changed by changes in the state of the button. For each frame, we update two variables `wasPressed` and `isPressed`. `wasPressed` holds the previous state of the button while `isPressed` holds the current state of the button.

TABLE I
State Changes

| | | |
|---|---|---|
| 1 | !wasPressed&& isPressed | just clicked |
| 2 | wasPressed && isPressed | holding down |
| 3 | wasPressed && !isPressed | just released |

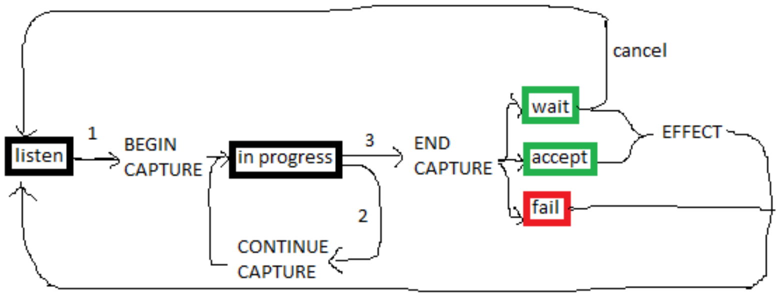


Fig. 8. Diagram of the state machine

### A. API

This state machine is managed by a class called GestureSystem. Gesture system is initialized by a constructor that takes `ZScore` as an argument. And as shown in the diagram, the calls to GestureSystem are `BeginCapture`, `ContinueCapture`, and `EndCapture`. These methods manage the data collection and take a Tranform as a parameter. `EndCapture` will output a `Dictionary<GestureClass,double>` that contains all the output layer values from the network. The user can choose the next step after having this information. Our interface provides an additional function`OutToGesture` which takes this dictionary and outputs the Gesture with highest probability or returns GestureClass.Unknown if none of the probabilities are larger than some threshold.

An optional parameter of the GestureSystem constructor takes a MonoBehavior of type `Visualize`. The class is created by the API user and contains an interface for 3 elements. The first two are an axis and a plane. These represent the normal axis and the plane (computed by the preprocessing) on which the gesture is drawn. These elements are mostly for debugging purposes. The third element is a LineRenderer which represents the trail of the gesture, which can be modified to fit the users needs. The `Visualize` class, if passed as a parameter to the constructor, is automatically called by the API functions to draw the chosen elements.

This state machine provides the backend of an easy to use API for users to implement gesture recognition with great flexibility for the user to modify the interface to their liking.

## X. Interaction Interface

The main purpose of this new method of interaction, as described in Section I, is to move away from Discrete I/O's and into a more natural methods of interaction . Therefore in designing the interface, we wanted to keep it free of clutter: buttons, menus, and other peripherals. Our main method of visual feedback to a user is through colors.

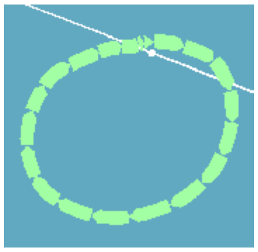


Fig. 9. The trail of a gesture will fade to green if recognized

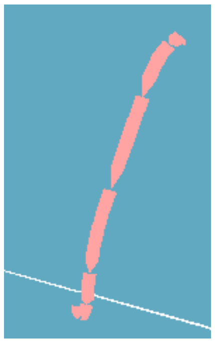


Fig. 10. The trail of a gesture will fade to red if not recognized

This simple and intuitive feedback is easy to recognize and doesn't clutter the interface with text or dialogs.

Using this feedback system, I was able to design an interface built for interacting with 3D models in and intuitive and natural way. Below I have listed what effect each gesture is mapped to within my interface.

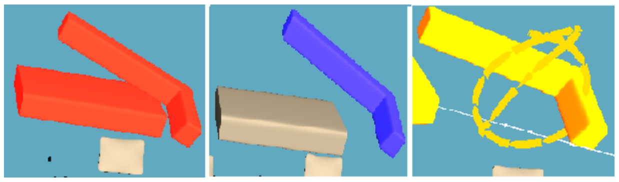


Fig. 11. Feedback for selecting objects, copying an object, and deleting an object, respectively

### A. *Gesture Effects*

- **Circle** - Select Group of Objects
- **Triangle** - Scale Up (works on group)
- **Square** - Fit to square
- **Upside Down Triangle** - Scale Down (works on group)
- **C** - Copy (works on group)
- **P** - Past (works on group)
- **Z** - Toggle Help Menu
- **S** - Recover previously deleted items
- **Delete** - Deletes objects (works on group) [This gesture can be canceled within 1 second]
- **Plus** - Adds/Removes from group
- **Left Swipe** - Rotates left (angle dependent on length of swipe)
- **Right Swipe** - Copy (angle dependent on length of swipe)
- **Infinity** - Locks the object at its local center and allows for free rotation
- **Tornado** - Spins object in place to give you full 360 view

## XI. Conclusion

Gestures are powerful in the realm of Virtual Reality as they are a natural method of interaction. And, in a Virtual Environment, a natural and immersive interaction is what users are really looking for. I believe that the integration of this interface can be useful in many fields from medicine to education. But, there are two fields that I believe are of great value. I believe that the integration of this interface can be useful in many fields from medicine to education.

For example, the interface would be applicable to 3D modeling software. Today, engineers, designers and many other professionals rely on 3D modeling software like Maya, Solidworks, CAD, etc. each and every day. But most of this modeling software is in 2D. An interaction with objects in 3D would be the next step for these modeling softwares.

I hope that my project lays the stepping stone for a future advance in Virtual Reality technologies toward more immersive experience.

## References

[1] Michael A. Nielsen, ”Neural Networks and Deep Learning”, Determination Press, 2015
[2] Bowman, Doug A. 3D user interfaces : theory and practice. Boston: Addison-Wesley, 2005. Print.