

6.835 Intelligent Multimodal User Interfaces
Massachusetts Institute of Technology

Author:
Zackary Anderson

Abstract

This paper presents a new paradigm for “time-based-information” interactions. The system, called a WallComputer, provides similar ease of use and the rapid accessibility of devices such as calendars and clocks, while also providing a rich and extensive set of interactions generally only found on a multipurpose PC. The presented system incorporates two input modalities: gesture and speech. I present the design of a gesture recognition system that is able to achieve over 99% accuracy, and the design of a contextual-booster that dynamically restricts the command set of each modality. Finally, I demonstrate that dynamic command-set-restrictions allow the system to achieve superior performance.

May 15, 2009
## Contents

1 Introduction ................................. 1
   1.1 The Task .................................. 1
   1.2 Related Work ............................... 2

2 System Architecture ......................... 3
   2.1 Gesture Recognition Engine ............... 3
      2.1.1 Nearest Neighbor Classification ........... 5
      2.1.2 Transforming Normalization ................. 5
   2.2 Speech Recognition Engine ................. 7
   2.3 State-Machine/Contextual Booster .......... 7
   2.4 User Interface .............................. 8

3 Performance ................................ 8
   3.1 Gesture Recognition ....................... 8
   3.2 Speech Recognition .......................... 10
   3.3 Usability ................................ 11

4 Limitations and Future Work ................. 12

5 Contributions ............................... 13

6 Appendix .................................. 13
   6.1 Using the System ............................ 14
   6.2 Using the Gesture Recognition Library ....... 15
Figure 1: Current information-delivery devices fall under two disparate classes. One the one hand are highly-efficient domain-specific devices, and on the other is the versatile but less-efficient PC.

1 Introduction

Presently, there are two major types of electronic information systems. Single purpose systems such as a clock, home weather station, or calendar/planner are easy and efficient to use, but are limited to their domain. On the other hand, general purpose computers allow users to perform a number of tasks, but require a greater level of involvement (the act of sitting down at a computer, booting up, using the mouse, typing on the keyboard, etc.). This paper presents a new paradigm that bridges the gap between these two types of systems - the WallComputer. The WallComputer is a wall-hanging machine that exploits multiple modalities to achieve desired operability. The aim of this system was to make a highly usable, efficient, and casual interface to deliver time-sensitive information (such as a schedule, weather, stock quotes, and news) to the user.

A great deal of prior research demonstrates how different modalities and interface paradigms than the standard window, icon, menu, pointing device (WIMP) can improve efficiency. The WallComputer was designed with casual use in mind. For example, a user should be able to check his schedule in the morning while brushing his teeth, or quickly check top news headlines when he walks into his house after work. To attain these goals, the WallComputer uses gesture and speech modalities for user interaction, completely moving away from the WIMP paradigm.

In order to achieve high recognition accuracy across modalities, the system heavily exploits dynamically-restricted recognition engines. System design revolves around the following hypothesis: when possibilities are constrained, performance rises.

1.1 The Task

The system presented in this paper is a wall-hanging computer that is void of windows, a pointing device, or a keyboard. Instead, it uses a clock-like UI-metaphor to display
Figure 2: WallComputer interface consists of a circular LCD-panel viewing area, a gesture pad, and a microphone. The steel frame makes the system wall-hangable. The user interface consists of multiple use-modes including weather and news.

information on a circular screen. A user interacts with the system by gesturing on a touch pad and talking to it. To demonstrate the capabilities of such a system, a prototype was built that displays both weather and news.

1.2 Related Work

There are commercial products that attack a similar user pain-point. For example, the Chumby is a small device with an LCD screen that can display RSS news feeds, weather, and a photo album. It is designed to be a casual, almost ambient, information source in the home. Similarly, there is a startup in the Boston area called Tap ‘n Tap that is building a similar product on top of the Android operating system.

There is extensive prior work in each of the input modalities used in the WallComputer. The gesture recognition system uses a variation on a nearest neighbors algorithm, and the speech recognition engine is a third-party library. This paper is not the first to propose dynamically-restricted command sets as a performance-boosting methodology. For example, Young, et. al. demonstrate that contextual inference can dramatically
improve speech recognition performance\textsuperscript{1}.

The proposed system unifies much of this previous work. It joins an emerging device (an ambient information source like the Chumby) with the power and efficiency of multiple interaction modalities, tied together with a contextual booster that dynamically restricts command sets for superior performance.

2 System Architecture

The WallComputer was designed with several goals in mind. The primary goals that drove the design process are:

- Efficiency: allow users to very rapidly access information.
- Casual Use: enable users to easily interact with the system even while standing and performing some other task.
- User Satisfaction: ensure that recognition accuracy and UI design are such that the system is perceived as pleasing to use.

In order to ensure outstanding recognition accuracy, the system dynamically restricts recognition possibilities in each of the input modalities. This is based on the current context (mode) of the system. As an example, when the system is displaying a user’s schedule, it does not need to recognize voice commands and gestures only used when displaying news stories.

2.1 Gesture Recognition Engine

The WallComputer gesture recognition system was built from the ground up. It uses a custom nearest neighbors classification algorithm using weighted Euclidean distance measures on normalized and transformed input, and a dynamically-restricted gesture set.

To read gestures, the WallComputer uses a USB touch pad mouse. While the desired output from the device is change in x and y coordinates, all that is available from the operating system, without rewriting the device driver, is current screen position of the mouse. Simply reading this position as a user gestures on the touch pad is unacceptable because of corner effects. If the mouse is against an edge of the computer screen, it will not move any further, even if the user is still moving across the touch pad. To work around this, the gesture recognition engine resets the mouse position to the center of the

\textsuperscript{1}“High level knowledge sources in usable speech recognition systems,” Communications of the ACM, February 1989.
Figure 3: WallComputer system architecture. Two input recognizers (one for each modality) run in the background listening for user input. A user interface displays information to the user. The State-Machine/Contextual Booster ties all three components together and dynamically updates the restricted command set in each of the two input recognizers.

Figure 4: The WallComputer gesture recognition engine dynamically restricts the possible gesture set depending on current system mode to improve recognition accuracy. In this example, we see the total 10-gesture set. In red are the four valid gestures while in “news” mode. Upon switching to news mode, the State-Machine/Contextual Booster notifies the gesture recognizer to restrict its recognition to only the valid commands for that mode.
screen at a predetermined refresh rate, and then reads how far the mouse moved within a time interval. From this data, the gesture recognition engine can extract velocity data while avoiding edge effects. The key is to set the refresh rate faster than a user can hit the edge of the screen. This system effectively disables use of the mouse, which is not a problem in the context of the WallComputer since no pointing device is ever used.

2.1.1 Nearest Neighbor Classification

The gesture recognition engine uses a nearest neighbor technique for unknown gesture classification. When a new gesture comes in, the algorithm computes the distance of each state vector along the gesture’s path to that of each gesture in a subset of the training set. The training set subset contains all examples of trained gestures that are of types included in the current restricted command set. The system then classifies the gesture with the label of the training set gesture that was closest to the input. Several implementation details were necessary for this high-level idea to perform well.

First, a four-dimensional state vector is computed from the velocity data collected using the above mentioned pointing-device-reset methodology. The state vector consists of the velocities as well as x and y coordinates computed by numerically integrating the velocity data.

Second, the unknown gesture input is normalized to reduce the effects of features that are generally insignificant such as gesture size. The next section describes this step in greater detail.

Next, for every gesture in the training set of a type contained in the current restricted gesture set, the Euclidean distance from the unknown gesture to the training gesture is computed. The Euclidean distance is weighted. Before taking the distance measure, each state vector is multiplied with an empirically-derived bias vector to weigh the path more heavily than the velocity data.

The final classification is the gesture type of the closest example found in the training set.

2.1.2 Transforming Normalization

In order to effectively compare different gesture acquisitions with one another, the WallComputer employs a transforming, resampling, normalization algorithm. This 1) trans-

\[\text{Suppose the training set contains five labeled gestures: two are of type `figureseven' and the other three are of type `wshape,' `ushape,' and `left,' respectively. This training data consists of a list of 2-tuples, each containing normalized path data and a label. If the current restricted gesture set is a list containing just the `figureseven' and `ushape' labels (this set only contains string label names), then an incoming gesture will be compared only to three training examples: the two `figureseven' examples and the one `ushape' example.}\]
forms gestures so that temporally-similar gestures have similar output state vectors [see Figure 5], and 2) resamples the path to a constant number of sample points. The objective of this normalization is to facilitate gesture comparisons.

Resampling occurs first. Paths are either downsampled if there are more than the constant number of sample vectors (the system works very well with this constant set to 20), or they are upsampled if the acquisition contains less than the constant number of sample vectors. Downsampling occurs by uniformly selecting the constant number of sample vectors from the acquisition. Upsampling occurs by uniformly repeating sample vectors.

After resampling, gestures are transformed to a new coordinate system and scaled to precisely fit within a normalization box. The WallComputer uses a box size of 100 pixels. Aspect ratio is always maintained - the gesture is scaled to fit the largest dimension.
2.2 Speech Recognition Engine

The WallComputer uses the Microsoft Speech SDK for speech recognition. A custom wrapper was developed to allow dynamic grammar changes at runtime. The Python wrapper spawns a speech recognition thread that waits listening for a user command. The State-Machine/Contextual Booster (SMCB) updates the grammar of the speech recognizer whenever a state change occurs (state changes might be the result of input from another modality, for example, a flick gesture). When a command is recognized, it puts the command name in an output queue. The SMCB continually polls the speech recognition object running in the background for an update. If there is a pending command in the queue, the SMCB operates upon it.

2.3 State-Machine/Contextual Booster

The State-Machine/Contextual Booster (SMCB) ties all of the WallComputer components together. It spawns the gesture recognition engine thread and the speech recognition thread. It then polls both of these for updates. If either modality has user input, the SMCB notifies the user interface of the input. This SMCB/UI interface was initially going to occur via remote procedure calls (RPC), but due to implementation complications, I opted for a simpler but far less robust shared file communication structure.

Shared file communication works as follows. A shared file is written to by only the single-threaded SMCB, and the user interface process reads the file as read-only. Since communication is one-way, the SMCB has to be able to communicate a repeated command without any form of acknowledgement from the user interface. To demonstrate this issue, imagine that the user issues a clockwise rotation gesture. The SMCB writes command=clockwise to the shared file. Now the user performs a second clockwise rotation gesture. A trivial implementation would lead to no viewable change in the shared file, meaning the user interface does not see a second issuance of the same command. To work around this problem, a sequential issuance of the same gesture or verbal phrase alternates between two different commands. For example, the first write would be command=clockwise and upon the second clockwise rotation gesture, command=clockwise2 would be written. If a third clockwise rotation gesture is input, it can write command=clockwise. This ensures that there is always a transition, even with repeated commands. This strategy can be compared to coding methods used in wireless communications such as Manchester encoding. Due to the one-way nature of the interaction, both the SMCB and the user interface need to store state of the current system mode. While the presented communication strategy works, it is not the most elegant solution.
2.4 User Interface

The WallComputer user interface is a Flash/ActionScript-based “circular interface.” The design of the user interface was inspired by the face of a clock. All of the mode widgets are designed with a circular layout. The weather mode has a circular sector that closes to form a circle and then opens to show current weather status. The news mode consists of eight blobs arranged about a circle that each contain a headline. The user can rotate these blobs into a “holster” that displays the story text in the center. See Figure 2 to see the news mode interface.

The basic architecture of the user interface is a Flash/ActionScript project with several frames that each represent a mode, and a utility that polls the communication interface (from the SMCB) for updates. If a command comes in, the user interface makes the appropriate updates. For example, a left flick may switch from weather to news, and a clockwise gesture while in news mode will rotate the news blobs to load the next story.

3 Performance

Performance tests demonstrate that the presented multimodal system achieves high recognition accuracy by coupling different modalities and context-based dynamically-restricted command sets.

3.1 Gesture Recognition

The gesture recognition engine is able to achieve a high degree of accuracy with minimal training. To evaluate performance, six different test subjects were asked to input on the touch pad five examples each of ten gestures. The ten gestures were presented to the subjects on a piece of paper. See Figure 8 for the tested gesture set. The labeled data collected during these examples was recorded and stored for evaluation. For each person, the system was then trained on the first example from each gesture. Then performance was evaluated by testing the pool of the remaining four examples per gesture (40 total per person). The labels on these examples were ignored - the system classified them blindly and then checked the classification against the label. This process was then repeated with only five out of the ten gestures collected. Since there were 5 examples of each gesture, one of which is used for training, this second experiment tested 20 gestures per person. These tasks were performed for each test subject, and average recognition accuracies were computed across all individuals. Figure 7 summarizes the results of this study.

Figure 7 demonstrates that restricting the gesture set improves performance, which validates a key motivation behind the WallComputer system architecture. Figure 9
Figure 7: Gesture recognition accuracy with different gesture set sizes. By minimizing the number of gestures in the current set, recognition error rate fell by 100%. This data was collected based on a system trained on a per-person basis with just a single training example per gesture. For a link to the collected data of 300 gestures, see the Appendix.

![Recognition Accuracy](image)

Figure 8: Gesture set used for testing. Recognition is heavily dependent on the tested gesture set. If very similar gestures were included in the gesture set, it is probable that the recognition accuracies given in Figure 7 would be lower.
Figure 9: Gesture recognition accuracy with varying training set sizes. More training was generally beneficial for the system, boosting accuracy. However, the system was able to achieve high (>99%) accuracy based on just one training example per gesture from each user.

demonstrates system performance with varying training set sizes. For this experiment, the entire 10-gesture set was used with the first $n$ examples of each gesture used for training, and the remaining $5 - n$ examples were blindly classified to evaluate performance. Importantly, the chart demonstrates that large training sets are not necessary for high-performance. In fact, an outlier in the training set has the potential to heavily degrade system performance because the implemented algorithm looks for the closest match. In addition, a system with a larger training set requires exponentially more time to classify user input.

While not rigorously tested, the system trained on one individual was still quite accurate in classifying the same gestures of another individual. Much of the person-to-person variation was in factors such as scale, which the gesture recognition algorithm automatically adjusts for.

### 3.2 Speech Recognition

Speech recognition was performed using the Microsoft Speech SDK, a mature, freely-downloadable speech recognition and synthesis package. Since there are papers that analyze the performance of this engine, less extensive testing was conducted on the speech recognition component of the WallComputer. The primary goal of performance evaluation was to see how a restricted grammar affects accuracy.

Tests were conducted with five different grammar sizes, ranging from two to thirty-two commands. This range represents a reasonable range for a WallComputer interface. The custom python wrapper dynamically loaded different grammars into the speech engine. These grammars consisted of real words used in the WallComputer system. Where possible, grammars for each test set were kept within a domain to simulate actual operation.
Figure 10: Speech recognition accuracy with different grammar sizes. The Microsoft Speech SDK is able to achieve nearly 98% accuracy as long as the command set is small and words do not sound similar.

For example, a calendar mode in the WallComputer (not yet implemented) might expect a day of the week or month to be said. The system was trained by reading passages to the computer prior to testing. Figure 10 demonstrates that the error margin drops off by 50% as the command set is reduced from 32 to 16 words. As long as dissimilar sounding words were in the grammar, command sets below 16 words had roughly similar performance with only about 2-3% false recognitions.

It should be noted that testing was conducted in a quite, controlled environment with a microphone close to the speaker. Ambient noise, microphone distance, and user training all have significant effects on performance.

3.3 Usability

Overall, test subjects enjoyed using the WallComputer. Users said the system is “quick and easy to use,” “simple to learn,” and “fun.” Even though testing was conducted on a standard computer using peripherals (a touch pad and microphone - not the finalized WallComputer interface shown in Figure 2 and Figure 11), users still found the system easy to walk up to and then casually and quickly interact with.

Negative remarks on system performance included the slow response time and the lack of tactile feedback. The system typically takes one second to react to user input, which is far too long for the reaction to appear seamless. The long response time sometimes caused users to repeat commands, thinking the system did not understand the gesture/voice-command the first time. One user requested that there be more tactile feedback in the
UI. For example, when “flipping” between different modes using the finger flick gesture, the UI widget should flick across the screen, not just change.

The existence of the gesture pad was both a strength and a weakness. For “command” gestures (such as the alpha gesture in Figure 8), having a separate gesture pad intuitively made sense to users. However, for manipulating gestures such as the finger flick, which aims at directly manipulating the screen, users felt that the gestures should be delivered by touching the object on the screen. For these types of gestures, a more expensive touch screen would have been better. Since both a touchscreen and the touch pad act as mice, such a system change would be completely compatible with the current code base.

4 Limitations and Future Work

Several important limitations were presented in the previous section, including the lack of tactile feedback, the sometimes-awkward use of a gesture pad to manipulate screen widgets, and slow response time. To address these issues, I plan on making UI tweaks to provide visual feedback of direct manipulation gestures, to perhaps integrate a touchscreen that can be used in conjunction with or instead of the gesture pad, and to migrate to using remote procedure calls for the Python to ActionScript interface. The majority of the lag time the system exhibits is due to the hastily-engineered communication protocol.

While system recognition accuracy was quite high, even one or two false recognitions out of every hundred are seen as annoying to a user. Thus, it is always the goal of the system designer to improve recognition accuracy. An idea on how to do this with the gesture recognition engine is by exploiting salient features, instead of dumb distance-measuring. The current gesture recognition engine has no concept of salient features. Research in human psychology shows that certain features in an image are more salient than others\(^3\). A possible improvement to the gesture recognition system would be to weigh the existence of salient features more heavily. For example, the presence of a sharp corner, such as that seen in the figure-seven from the training set in Figure 8. Another possible way to make the gesture recognition more robust is to use an average nearest neighbors technique. As of now, an outlier training example can throw off recognition for the whole system. By classifying based on minimization of the average distance to all examples of a given gesture type instead of just the closest, the system will be less prone to outliers in training.

\(^3\)For example, the Gestalt Principles detail several visual features people use as salient for grouping objects.
5 Contributions

I have presented a system that couples two modalities and exploits contextual information (the mode of the system) to boost performance. I demonstrated that by dynamically restricting the command set, the system is able to achieve extremely high accuracy rates. These are the key contributions in this paper:

1. Designed an accurate (>99%) gesture recognition system based on optimizations of a nearest-neighbors algorithm. The library is decoupled from the rest of the WallComputer, allowing for re-use in other systems.

2. Demonstrated that multimodal, contextually-restricted user interfaces provide superior performance to single-modality, non-dynamically restrictive systems.

3. Presented a new paradigm of computer interaction that verges between ambient and full-PC capability.

4. Built a functional “WallComputer” with several modes of operation.

6 Appendix

All materials developed for this project can be found at:
http://web.mit.edu/zacka/Public/wallcomputer/

In the /software folder there are five source files:

mousegestures.py is the gesture recognition library I wrote.

speechinterface.py is an extension I wrote to a third-party Microsoft Speech SDK Python wrapper (speech.py).

wallcomputer.py is the state-machine contextual booster I created to tie the two modality threads with the user interface.

WallComputerUI.fla is the ActionScript/Flash user interface I wrote to display information to the user.

drawWedge.as is a third-party drawing library used for some of the animations seen in the user interface.

All data analyzed in the Performance section (the 300 gesture examples) can be found and independently analyzed in the /test_data folder. Training data I use by default is
in the /zack_training_data folder. Miscellaneous gesture image exports from mousegestures.py can be found in /saved_gesture_images. My 6.835 final presentation can be found in /presentation. An electronic version of this report can be found in /report.

6.1 Using the System

To use the system, navigate to the software folder and open WallComputerUI.swf in Adobe flash player (or a web browser). Then use Python to run wallcomputer.py. This will activate two listener threads (one for voice recognition and one for gesture recognition), the SMCB, and the flash-based user interface.

While the system requires some third-party python modules (such as pylab for using the built-in advanced debugging routines), these can be freely acquired online. The system was written in Python and ActionScript to allow for cross-platform compatibility. Unfortunately, the Microsoft Speech SDK only runs on Windows operating systems, however.
6.2 Using the Gesture Recognition Library

The gesture recognition library was specifically-designed for re-use in other software projects. There are several ways to use the system. Here is a quick overview of useful commands:

**10-Second Quick Start:** Print out recognized gestures.

```
m = MouseGestures()
m.listenForGesture(2, 'firstgesture')
m.listenForGesture(2, 'secondgesture')
m.saveTrainSet('mytraindata.py')
m.listenForGesture()
```

The first command instantiates a MouseGestures object. The second and third commands tell the system to train two examples each of a gesture named ‘firstgesture’ and a gesture named ‘secondgesture’. A user can select the name and number of examples to train each gesture on. As many gestures as desired can be loaded into the system. The saveTrainSet() method call is optional here, but allows the user to store the training data just acquired for later retrieval. This saves the time of training the system again. The last command sits in a loop until the user exits. This loop listens for gestures on the touch pad and prints out the name of the recognized gesture.

**Use Restricted Gesture Sets:** Spawn a gesture recognition thread and dynamically change the gestures.

```
m = MouseGestures(headless=1, trainingSet='traintest')
m.serialSetRestrictedGestures(['gesture0', 'gesture1'])
m.lastGesture()
m.kill()
```

This set of commands will make a new MouseGestures object, dynamically load in a new gesture set, and then wait until a gesture is recognized and print it. Try out a gesture before or after issuing the m.lastGesture() command. The kill command returns the mouse to user control, terminating the recognition engine. For a non-blocking read, one can use m.lastGestureNoBlock(). This returns -1 when there is no gesture in the queue.

The system also includes several methods for training the system, loading in batch data, and conducting/scoring performance tests. See the code and inline comments for more info. There is also a rich set of debugging capabilities if the debug variable is set to 1. For example, the system can draw the normalized user input gestures to an image file.
The gesture recognition library uses an operating system hack to read gestures from a USB touch pad. The code currently supports both Windows and Linux operating systems.