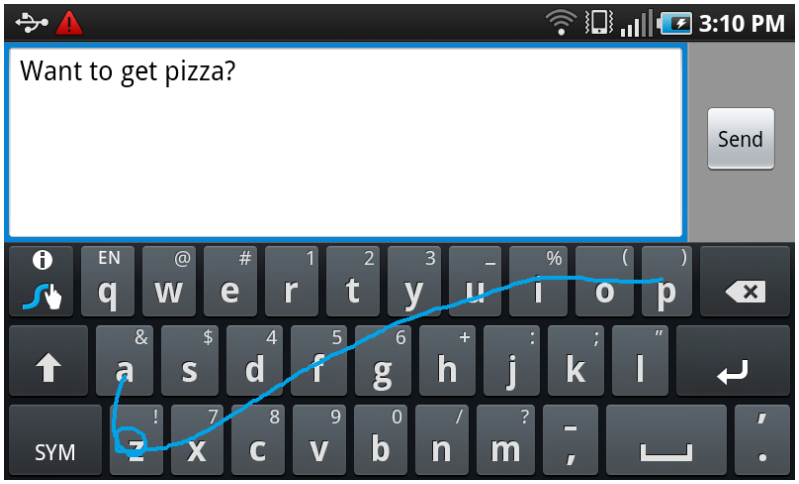
Swype in the Air Proposal

Michael Zhong (mqz2001)
Word Count: 1,795

Overview:

For this project I would like to implement a system that can enter text by making Swype-like gestures in the air. Swype is a text-entry technology offered on the Android that, by using one continuous gesture with your finger, can input text on a per-word basis. The gesture is defined as such: move your finger over each character in the word you want to generate in one continuous gesture, then release your finger from the touchscreen.

Example of Swype (via linkeduplife.com):



My project aims to implement this exact same concept, but by allowing the gesture to be performed in the air rather than having the gesture be fixed to a touchscreen. Swype over the last few years has looked into doing this, and they currently are in the midst of developing the technology for it, but they haven't produced an in-the-air product yet. For full disclosure, in my Mobile HCI course, I'm doing a similar project—for that project I'm implementing and surveying a series of simpler (i.e. not the Swype technology) in-the-air text-entry

technologies and comparing the performance of these simpler text-entry systems with each other.

To my knowledge, there is no existing per-word text-entry system that's implemented through the air. There are per-word text-entry technologies (SHARK, Swype, SwiftKey, Android Gesture, among others), and there are text-entry systems through the air (but they enter text on a *per-character* basis), but none that combine both. This project aims to delve into that field.

This project should be considered to be an example of a visual interface. It takes in a visual input (from a Leap Motion device—discussed in class), and by processing this visual input, produces an output. In many ways it is similar to the Visual Password assignment we implemented. A visual input (a hand gesture) is received, and an output is produced.

Limitations:

According to the Swype website, the product is the result of 10 years worth of development work. I don't have that long to complete this, nor do I have a staff of full-time professionals. Therefore, naturally, my project will have limitations.

First, should implementing a text-entry system in the air prove to be too difficult, a logical bailout step (listed under the Steps section below) would be to change it so that an input is generated on a flat surface (e.g. using a mouse to draw a gestural pattern instead). Second, another limitation I'm imposing is that this project is with the evaluation/performance of this project. I've placed much lower evaluation standards (elaborated more below but in short the performance is evaluated by 1) text-entry speed, 2) error rate) on this project relative to Swype. According to Swype, most of the development time was split between developing the algorithms and fine-tuning the performance. The algorithms for per-word text-entry on a flat surface (though not Swype's) are publicly available, and I'm not

going to concern myself with coming near Swype's performance, so I believe this project is manageable. Third, related to fine-tuning performance, I do not intend to implement some of the features Swype has. Swype has suffix completion, ways to gracefully exit, and real-time display list word generation, among other features. This project will implement none of those concepts.

As far as special equipment goes, I intend to use a Leap Motion device (already acquired) to process the hand gestures. The Leap has built in finger and hand detection libraries which I believe will speed up development.

Steps:

The following steps closely model the pipeline outlined in Swype's patent for its technology.

Step 1: Preprocessing the dictionary of words

The first step is to acquire a database of words with corresponding frequencies. Swype has the ability to learn additional words and infer words based on the context of other words in a sentence. My project will have neither function. One logical way to acquire a database could be to simply use an existing collection (for example, the Penn Treebank or the Wall Street Journal Corpus). The domain for both corpuses however, is arguably not the same as the usual words the average person will type. To acquire a more colloquial database, I'll combine both existing corpuses (Penn Treebank and the Wall Street Journal Corpus), with some trivial web scraping to acquire a more diverse set of words. By combining both datasets, there will be a dictionary of words available for access.

Step 2: Preprocessing the input pattern

Working on a flat surface here (but unlike Swype, without the keyboard as a reference point), identify the gesture pattern and translate it. This means, that given a drawn path, identify 1) all the abrupt changes in the

path (i.e. when a letter has been selected) and record the corresponding (x, y) position, 2) the speed of the stroke and the angle to get from one position to another, 3) the start and release points of the path (trivial for a flat surface, considerably more difficult in the air (no tactile feedback). Also, Swype specifically incorporates a double letter pattern, and I will do the same here.

Step 3: Pattern Recognition Analysis

Now, given the (x, y) positions of all the entries (and whether a double letter pattern was associated with that particular position), the stroke/angle to get from one position to another, and the start and release points of the path, we can map this data to a series of potential words.

First, each (x, y) position will map to what it believes to be the most likely character, as well as surrounding characters with their respective probabilities. So for example, for the letter 's', s will correspond to a high probability with s (naturally), a lower but still higher probability with w,a,x,d,e (characters that neighbor s), and a near-zero probability to letters far away from s on the keyboard such as p or m. SwiftKey and Swype simply decrease probability with respect to distance, and I will do the same. Second, the angle and speed of the stroke will also be mapped to probabilities reflecting confidence.

Third, one big way this project differs from other text-entry systems is that there is no reference point via a physical keyboard. The beginning point could correspond to 26 different corresponding mappings to a keyboard. Therefore, I'll iterate over all of them.

Then, given all these probabilities, the program can then map out all possibilities in words. By incorporating the likelihood of different key functions, this algorithm can greatly reduce the complexity of the possible options, thus speeding up the program. Using my background in classes like NLP, Bayesian Analysis for NLP, and SLP, I'll then devise an approach (probably via Bayesian analysis) to generate the most

likely combinations. These will be ranked based on a score (I intend to use a scoring system similar to what is outline in the paper Hex: Dynamics and Probabilistic Text Entry (see references)) and then outputted.

Step 4: Creativity (Adding the "in the air" part)

Should Step 4 prove too difficult—although I intend to implement my own score weighting algorithms above, Step 4 is the most truly novel part—I'll deem Step 3 an acceptable bailout point. In this step, I'll map the flat surface input and adapt that for the in the air component.

First, this means identifying the finger and tracing its path. The Leap Motion has a built-in library that I've already used that does exactly that. Second, a similar in-the-air gesture text-entry system (Unigest) flattens the 3D path to a 2D path. I will do the same using similar methodology. Neither step should be difficult.

The trickiest part I think will be recognizing when the gesture begins and when the gesture ends. Should this prove too difficult, I will introduce another bailout point by manually indicating the beginning and end of a gesture. I have some ideas though for indicating gesture beginning and end. One approach could be to use the second hand to indicate a start and stop. While this would be the most accurate approach, it also is the most inconvenient, the slowest, and the most physically taxing. Alternatively, I'm thinking of getting the user to return his/her extended finger into his/her fist when done with the gesture. While quick and convenient, early experimentation with trying this with the Leap has indicated that the finger withdrawal itself can generate an undesired path. Factoring this unintended path out will be a challenge.

Once that's all done, the project will be complete. Performance will be measured on two criterion. First, how often is one of the top three most likely words the actual word intended to draw? That is, what is the error rate? Second, how quickly can the user type? I intend to

provide a set script of words to type, and time the time it takes to collectively swipe through everything.

Evaluating Results:

The standards for this project will be much, much lower than the professional Swype. While the user is sitting, a Swype-like interface (reference #8) achieved an error rate of just 2 percent and an entry speed of 25 words per minute. I will arbitrarily say that acceptable performance for me would be a 20 percent error rate with a slower swiping speed (15 words per minute instead) because the text entry is occurring through the air (and thus more taxing and slower).

I will run ten trials on two set scripts, each trial representing a different person. I expect to run into failures, and with these failures I believe I can learn where the program fails (for example, if the program misrepresented a stop point or if the program couldn't identify the finger or if the program attached irrational probabilities).

References:

1) Castellucci, Steven J., and I. Scott MacKenzie. "Unigest: text entry using three degrees of motion." CHI'08 Extended Abstracts on Human Factors in Computing Systems. ACM, 2008.

2) Kristensson, Per-Ola, and Shumin Zhai. "SHARK 2: a large vocabulary shorthand writing system for pen-based computers." Proceedings of the 17th annual ACM symposium on User interface software and technology. ACM, 2004.

3) Kristensson, Per-Ola, and Shumin Zhai. "Relaxing stylus typing precision by geometric pattern matching." Proceedings of the 10th international conference on Intelligent user interfaces. ACM, 2005.

4) http://www.extremetech.com/extreme/97837-how-does-swype-really-work

5) Kushler, Clifford A., and Randal J. Marsden. "System and method for continuous stroke word-based text input." U.S. Patent No. 7,453,439. 18 Nov. 2008.

6) Williamson, John, and Roderick Murray-Smith. "Hex: Dynamics and probabilistic text entry." Switching and Learning in Feedback Systems. Springer Berlin Heidelberg, 2005. 333-342.
7) https://www.leapmotion.com/developers
8) Yatani, Koji, and Khai N. Truong. "An evaluation of stylus-based text entry methods on handheld devices in stationary and mobile settings." Proceedings of the 9th international conference on Human computer interaction with mobile devices and services. ACM, 2007.