Spell Check

Denis Doçi, Neil Rao, Anthony Nedumgottil, Brandon Davis, Katarzyna Krawczyk

Introduction

Our project was initially designed to be a virtual reality based real-time spelling checker. In our original design, the user would wear a Microsoft HoloLens, and on their command they will be able to identify text, and check the spelling of the identified text. Initially we went through a multitude of project ideas, but most of them centered around the idea of text recognition and applying some functionality to the recognized text. We thought of recreating the Google Translate application in a virtual reality space so that the user would be able to translate foreign text into their native one as a helpful tool to help students and traveling tourists. This however just seemed to be a recreation of an existing tool, and we wanted to create something more original and innovative. This led us to the spell check idea.

In actual development, however, our design was not fully executed. This was due mainly with issues we were having with the HoloLens itself. We got a fully implemented backend, and all of our functionality was done. Porting all of this functionality to the HoloLens came with a lot of issues. To get a proof of concept, we launched a web application and created an Android Application. This served to prove that we could in fact accomplish all of our functionality in reading in text, analyzing the text, and doing spelling corrections and suggestions without the HoloLens itself. Implementing the Hololens came with many issues. We could not get the HoloLens to trigger gesture

events, calls to the API were returned with errors, and the HUD would be in constant motion. The project we imagined was intended for an augmented reality environment so this roadblock was very unfortunate. With the creation of the Android application, however, we do have the possibility to port the application to Android native AR devices such as the Vuzix or Google Glass.

With this application, we hope to target a large variety of people. We divided our target audience into three main groups: school children, non-native speakers, and people with reading/writing impairments. This division is based on the fact that the main goal of our project is to bolster the user's ability to write correctly. This command of language in written form while lead to better understanding of the language itself which would be beneficial to students learning English and foreign speakers. Also, it would be beneficial to the day-to-day lives of people with impairments such as dyslexia that have difficulty writing and spelling. That being said, this product can be used by anyone. Some sample use cases would be a professional recruiter. Recruiters often say that spelling is one of the first things they check on a resume, because it shows the professionalism of the applicant. In addition to this, someone like an English teacher who grades physical assignments with spelling being one of the criteria of their evaluation could also benefit from our product. There are many other examples of use cases for our design. With the use of our application, a normal person's ability to spell is greatly increased. We are therefore augmenting a person's natural writing abilities and augmenting their command of the English language.

Prior Works Research

Prior work that we found for this project that we will be usable for ours:

1. Denis

Processing Error Correction Algorithm using Google Online Spelling Suggestion

One of the potential problems that we're going to run into is the accuracy of our spell checker after we process our characters. With optical character recognition, it is likely that most of the characters that are read in won't be exact. This brings up issues of valid and accurate spelling corrections. In Youssef Bassil and Mohammad Alwani's paper on creating a processing error correction algorithm, they attempt to solve this exact problem of optical character recognition errors. They do this by not just using a dictionary, but instead a linguistic context-based correction algorithm. The benefits of such an algorithm for any optical character recognition project are plentiful. Not only does it allow for a buffer for imperfections when the processing is done, but it also allows for recognition of context spelling errors to be picked up which a normal dictionary-based checker might not.

The algorithm itself is reliant on Google Search's "did you mean" feature. The algorithm begins by inputting the recognized text images to a output text B. B is set up as five separate blocks for five separate words. Then, all blocks are submitted as a search query to Google's web search engine; if the search returns "did you mean: x" where x is the alternative spelling suggestion for a certain block within the sequence, then that

block is considered misspelled. The algorithm therefore is not reliant on any dictionary, just Google itself.

This seems like a good approach, however the usefulness of this algorithm for our project might be slim. The main issue is obviously that whatever uses this algorithm must be connected to the internet at all times. This is an issue because we want our application to have wide uses, not just when the user has internet connectivity. That being said, the concept of using a context-based spelling checker along with a dictionary one is something we will implement. This will increase the accuracy and precision of our product greatly.

Word Lens

This prior work is more of an analysis of an existing commercial product. Word Lens was an application formerly available on the Android and IoS market. This was an augmented reality based translation application. The user would take a picture of some text they wanted translated, and the application would provide an overlay of the text in the language they wanted it translated to. Many aspects of this application would be useful to ours. The most important being the OCR implementation. The application uses Google Cloud Vision API to recognize text and parse that text in a usable format. We used this in our web API as a direct result of researching this application. We did not need the translate function that Word Lens provided. But if you divide its functionality into recognizing text then translating it in a different language, we implemented the first part of this functionality.

2. Katarzyna

A Database for Handwritten Text Recognition Research

The article written by Jonathan Hull explains a system of handwritten text recognition using images that are stored on the cloud. The system is presented and used in a local post office. An image database was created with about 15000 images that included city names, zip codes, and state names. Each word has multiple handwritten styles so when the address is scanned it is easily recognized by the machine to be sorted. Creating such a database can be very useful when it comes to small amount of characters. Although this article sounds very encouraging it might fail when it comes to our project. Making a database for the whole dictionary would not work so well since it would need a lot of cloud storage. Every person has a unique handwriting making it extremely hard to recognize it. The idea seems to be still developing and just the fact that you would need that big of a data for just the addresses in a certain area makes this topic not that well developed yet. Although it might be very easy for humans to read most of the handwriting translating it to a computer or finding a way to be able for a computer to understand it gives a lot of thought and research.

This article has proved that reading handwritten text and recognizing it electronically can help in everyday life which what our project intended to do. Unfortunately coming to a realization that this might be not only too expensive to develop since we would need a large amount of data can store the text but it also would need a machine that would work with that database and be able to provide the correct result.

3. Brandon’s

“Using smartglasses for utility-meter reading”

The purpose of this research paper was to develop a way to automate data logging from utility- meters using smartglass. Due to the fact that these meters were mostly analog in nature, workers would have to handle data entry by hand which inevitably leads to mistakes and human-error. The idea would be to have a user use smartglass to first take a picture of a meter, use OCR to gain the data input, and then utilize cloud services to automatically log and store the data. The researchers decided to use the Vuzix M100, an android based smartglass that also has tactile buttons. In addition to this, they also explored using voice and gesture recognition, finding gesture recognition easier and more accurate (they noted the ability to use an infrared sensor helped greatly with this). The conclusion of their research was that they found relative success being able to scan, recognize, and store meter data with good accuracy. The work done in this research coincides a lot with what we want to do with their utilization of smartglass and OCR. Additionally, the Vuzix M100 is available to us to use, and knowing about their success with it is encouraging to know.

4. Anthony’s

Accuracy of Electronic Spell Checkers for Dyslexic Learners

This article talks about a study that was published for the Professional Association for Teachers and assessors of Students with Specific Learning Difficulties, or PATOSS, by

Abi James & EA Draffan talks about how modern day spell checkers work for correcting typos for people suffering from Dyslexia. First the two established that the most common reason why most spell checkers fail is because the lack of suggestions from a smaller dictionary. This reason applies to all users, not just those with dyslexia. Another reason why spell checkers fail for dyslexic people can be categorized by four different criterias: one letter wrong, one letter omitted, one letter inserted, and two adjacent letters transposed. These errors are a lot more common for dyslexic people, so now more modern day spell checkers changed their algorithm to accommodate and cover more of these issues by turning to a more phonetic approach. This approach checks to see how the word itself sounds and suggests correct words best on the sound. This approach is great as it has increased the amount of words corrected by spell checkers however, we have now learned that dyslexic learners also have trouble choosing the correct spelling from the suggested choices as they are not able to properly read them. These researches also conducted a test between 6 popular spell checkers and found that the spell checking tool, Franklin Literacy Word Bank, had the highest amount of corrected misspellings for the group of dyslexic users who participated in the study. Our plan is to use this spell checker to borrow ideas and features to help us target dyslexic users. However doing so might require a lot more effort and this might not be able to be completed within the scope of the class. While helping dyslexic students was one of our initial ideas for creating Spell Check, we might not be able to accurately check their spelling.

5. Neil’s

Recognizing handwritten text is more challenging than what initially meets the eye. Unlike recognizing specific fonts on printed paper (where each individual letter printed has the exact same curve), recognizing handwriting is a whole different ball game. What makes handwriting so much more difficult to analyze is two-fold. First, there is tremendous variance in handwriting styles, from print to cursive; every person appears to write in a somewhat unique fashion. Second, there is tremendous variance within a single piece of an individual’s writing itself; humans, being imperfect creatures, draw the curves of each letter slightly differently each time. While our complex biological pattern-recognizing brains can (through years of schooling) learn to easily differentiate and interpret these variant styles, teaching a computer how to do that involves an entire field of cutting-edge data science research. There are several popular approaches to digitizing text, all belonging to the general category of optical text recognition (OCR). The very obvious first set of algorithms I came across involved a template-matching approach 1 , where the geometry of characters were compared in an almost straightforward manner; these algorithms compared input characters to predefined templates, which meant that the characters were either recognized as an exact match or no match at all -- there was simply no room in the algorithms to account for variations such as tilts or writing styles. To try to accommodate these, another method was devised called “Recognition Using Correlation Coefficients” based on the cross correlation of input characters’ transforms, but this approach ended up being far too

tolerant (on the other end of the spectrum), causing the recognizer to erroneously mix up similar-looking characters like “B” and “8” or “O”, “Q”, and “0”. In a continued effort to overcome these problems, and as data science progressed, better methods were discovered, mainly involving the fields of machine learning. Various methods were devised involving everything from neural networks to hidden markov models. Regardless of the algorithms I found, they all followed the same general approach, albeit in different ways. At the highest level, you had a page of written text which was scanned. Upon scanning, usually the image would be processed into a grayscale image, dubbed “gray level image processing,” and from there the image would be segmented (i.e. spliced) into pieces where each piece contained no more than a single character. Then, certain *features* of the character would be identified (such as slope, stroke depth, etc.) and then extracted to form *feature vectors*, and from there be classified and then post-processed to combine them into words. TRIER et. al. summarized several existing pre-processing methods such as binarization / two-level thresholding, segmentation, and character representation conversion (skeleton format and contour curves) 2 . It went on to discuss various feature extraction methods, from “Zernike moments” to fourier descriptors, and onto classifiers using methods such as neural networks. There are many mathematical models to approaching this problem, and one of the most intriguing solutions I found was by Mathur et. al. which used not one neural network but several that were each tuned to slightly different writing styles then processed further by a genetic algorithm to select for the best output . Regardless of the specific algorithmic details, it became quickly evident that a machine-learning

approach was the way to go. However, none of the dozens of studies really focused on speed. Even the studies that discussed speed did so mainly in informal terms, using vague terminology that an algorithm can run in “moderate speeds” for example. It wasn’t that helpful, so we realize that in order to run an augmented spell checker in real time, we may need to spin up a few instances in the cloud to augment the hololens’ hardware capability itself. Then, it’d be a trivial matter to live-stream camera input to the server farm, process the input, and return simple metadata back to the hololens which would be processed extremely quickly. This adds the additional burden of needing internet access (as opposed to being a purely offline solution), unless the hololens turns out to be capable of handling the workload, which is only exactly possible to tell when actually testing with the device while coding our application. Regardless, we will ultimately settle on a single ML-based recognizer approach, but balancing out fast offline recognition with higher-quality (albeit slower) online recognition will take trial and error.

Citations

1. Bassil, Youssef, and Mohammad Alwani. "Post-Editing Error Correction Algorithm For Speech Recognition Using Bing Spelling Suggestion." International Journal of Advanced Computer Science and Applications 3.2 (2012): n. pag. Web.
   Accessible link: https://arxiv.org/pdf/1204.0191.pdf

2. Proceedings: Seventh International Conference on Document Analysis and Recognition: August 3 to 6, 2003, Edinburgh, Scotland. Los Alamitos, CA: IEEE Computer Society, 2003. Print.
   http://ieeexplore.ieee.org/abstract/document/291440

3. Vinod Chandra and R. Sudhakar, “Recent Developments in Artificial Neural Network Based Character Recognition: A Performance Study”, *IEEE, 1988*.

4. IVIND DUE TRIER, ANIL K. JAIN, and TORFINN TAXT, “FEATURE EXTRACTION METHODS FOR CHARACTER RECOGNITION | A SURVEY,” *MSU, 1995.*

5. Mathur, Shashank, and Vaibhav Aggarwal. Information Science and Computing(2008): n. pag. *OFFLINE HANDWRITING RECOGNITION USING GENETIC ALGORITHM.* Sixth International Conference on Information Research and Applications, 2008. Web. 2017.

6. “Using smartglasses for utility-meter reading”
Written by: Depari, A,; De Domincis, C.M.; Flammini, A.; Sisini, E.; Fassanotti, L.; Gritti, P. http://ieeexplore.ieee.org/abstract/document/7133649/?reload=true

7. Accuracy of Electronic Spell Checkers for Dyslexic Learners
https://www.dyslexic.com/accuracy-of-spell-checkers-for-dyslexic-learners/

Problems and Barriers

Problems we encountered were difficulties when it came to transferring programs and builds to the HoloLens when we have it available (such as builds that work on our phones/emulators not working the same when on the Microsoft HoloLens). Additionally, we found some difficulty when comes to character recognition with handwriting, especially when it comes to the handwritings of different users.

We also will have issues on suggesting the correct word depending on how severe the misspelling is. As we have learned, the severity of the misspelling can widely range for users who have dyslexia, therefore we might not be able to correctly spell check their words. One possible fix for this is to have some type of audio feedback on what the correct word should be.

What turned out to be one of the biggest barrier for us was the text recognition. Our original idea was to recognize the handwritten text live using Microsoft HoloLens. That way was not only ineffective because of the different handwriting but streaming Spell Check live from the HoloLens was impossible to achieve. We needed for the text to be still for the program to be able to underline the misspelled words. Using the HoloLens it would be much harder to implement it. Since if you were to wear the HoloLens, you would have to be very still because with every small movement the program would not be able to catch the text and spell check it. That is when we switched from the live approach to image recognition. Making that switch made us realized how difficult this project turned out to be and how expensive it can get very fast. Image recognition made it a lot easier for us to work with. This change made our application faster and more accurate when it comes to underline the misspelled words along with displaying the correct spelling.

Timeline

For this project we decided to go with a pseudo-Agile/Agile-lite method of creating this project. As a team we have made a trello board where we will create tasks for the project. Group members will work on tasks as needed and add more in. We call this an Agile-lite as we don't follow all of the traditional Agile methodologies laid out in the Agile Manifesto. Our sprints are done in 1 week rather than 2 so we can match the UIC class schedule. We don’t have daily Scrum meetings but rather will discuss in class and meet up in person on the weekends where we are all free. We don't fully felge out our tickets

on the scrum board as we want to focus on “Get it done and move on” and we want to adapt our project as we discover new barriers and problems. This timeline only covers working on the project itself and not on the paper or any other assignments for this course.

When we first started the Spell Check project, we had the initial idea of having a user experience where the user puts on their virtual reality device, looks at a document handwritten in the english language. While looking at the document, our Spell Check application will “spell check” the document in real time and then display a mistake mark, a red squiggly line, on the misspelled word and then the user would perform a gesture on the misspelled word to get an expanded list of suggested corrections. Now that was our intention, what we got finished at the end of the semester was very different.

Originally, the five of us came together and agreed on the idea above for our end goal project. We then split off and did research on how to build that idea into a product by the end of the semester.

We initially intended to use Google Glass as our virtual reality device and create our Spell Check application in Java, deploying it as an Android application. We decided on this because most of us were familiar with Java and creating applications for the Android platform, so the conversion to a Google Glass Android application shouldn’t have been too much work. After our first presentation, it was brought to our attention that it would be awkward for the user to use our real time spell checking with Google Glass. This was because Google Glass only has one tiny LCD in the right hand corner of the display. This would mean that they would have to constantly look in the corner

and the amount of information displayed to the user will be limited as it has such a small LCD.

After this discovery, we scrapped our plans of using Google Glass and tried to find another virtual reality device with a fully immersive display. We initially found the Vuzix m100 to be really appealing; it has two LCD screens that cover both of your eyes and also runs on Android. Sadly there wasn't too much documentation on the Vuzix for us to feel confident in our skills in creating our Spell Check application on the Vuzix and our search for another virtual reality device continued.

After more research, we came to the conclusion that the Microsoft Hololens would be the best choice for our virtual reality device to build our Spell Check application for. It has a full 2.3 megapixel widescreen stereoscopic display, so we can display information to the user in their full range of vision. The HoloLens runs a version of Windows 10 called Windows Mixed Reality, so we thought developing applications for HoloLens would be simple since we're all somewhat familiar with working with C#. Some group members had some familiarity with Unity3D so we thought the workload for the project wouldn't be too bad, so we aimed high and decided to do development right away.

We first started development of our neural network for handwritten text recognition. We researched many different neural network training libraries, we decided

to use the UJI Pen Characters Dataset Version 2 hosted and documentation by the University of California at Irvine. This dataset contains samples from 60 different writers from 2 different locations with 2 different phases, this allows for a wider range of samples in handwriting because humans don't write letters in the exact same way every time, there are slight variations and taking samples at different phases allows us to capture these slight variations better. The UJI Pen dataset contains 66 different letters, 10 numbers, and 21 special characters, like punctuation marks, for a total of 1364 unique characters. We thought this was going to be a large enough dataset for our application so we began training our neural network with this. We created a 900-900-8 neural network and began training. We created a C# application where we fed in images of single letters that we drew and started the learning process. While we were training it we found that even with the 1364 unique characters, we were still only able to recognize only <10% of the words from our handwritten samples. We tried many other OCR neural networks such as Tensorflow but we could not get them to work. We would need to have a lot more samples and a lot more time to train our neural network that what was permitted in the span of one semester, so we looked to other enterprises services to speed up development.

Afterwards we looked at using Microsoft's Azure Cognitive Web Services for our neural network backend. We got it to work with our C# application and we able to create a webservice that takes in an image and able to return a list of misspelled words. We also decided to use the Microsoft Bing Spell Check Api for our spell checking backend rather than developing one ourselves. We tried out a few sample texts that we

wrote with our new OCR engine and it seemed to detect the words correctly, our web service that we wrote then returned us a list of misspelled words that we could later use to display to the user. However things started to go wrong when we tried it out on a larger, more natural, sample text. We took some class notes, what the typical use case for most users of Spell Check would use as an input, and tried it with our Azure Cognitive Web Services version and we were able to detect almost nothing. We were able to detect less than <3% of the words on the page, let alone spell them correctly. The problem was that when we used our first sample text we used grid paper with very dark ink and spaced out our letters very carefully, now it's great that it was able to be detected but it's very uncommon to come across text like this in the real world, thus rendering this version pretty useless. Another problem we had was that the Bing Spell Check Api often miss read words and marked correctly spelled words as incorrect. This gave us a lot of false positives back for when we were displaying misspelled words to the user.

We were met with quite a few issues when it came to developing on the Hololens. In addition to the extensive set up process of downloading and acquiring the appropriate IDE's, packages, and emulators, making sure that all this software worked with each other (differences in versions, new updates not present in tutorials, etc.) was also a bother. Another major issue that we had to face was actually getting a build to deploy on the actual Hololens. While it was "simple" enough to port our work to the emulator, our development machines lacked the permissions to directly deploy our builds to the device. Instead we had to use computers within the university that already

had said permissions. This, while still a solution, was still troublesome considering we had to share the use of the computer and the Hololens not only with other groups using them, but with the researchers within the university who used those devices themselves. Because of this, we did most of our development work on the Hololens emulator. Debugging was very tedious though, as the process from getting a build from Unity to the emulator could take anywhere from 5 to 20 minutes depending on if relaunching the emulator itself was necessary. Simply put, the debugging process, even for fairly simple problems, was very lengthy.

The other big issue we faced was getting what we had to work appropriately on the emulator. The goal was to have a basic HUD detailing information to user and to have the user take a picture which would be sent to our web API to be spell-checked. Creating and getting the HUD to work was no issue, but we had the problem that the Hololens emulator has no access to its computer's webcam. Therefore we had no way of getting it to take a picture outside of its environment. To compensate for this, we tried placing a image of text within the environment for it to take a screenshot and send an image that way. However, the picture quality needs to be quite high for our web API to return any meaningful data - the images taken through the emulator environment were a little blurry at best.

After all these hurdles we were strapped for time, we didn't have much time. We were recommended to use Google Cloud Vision API for our OCR engine as it recognized words a lot better than Microsoft Azure Cognitive Web Services. We created

a Google Cloud instance and then began work on using this API with our C# webservice.

After not being able to deploy our initial application on the HoloLens, we had to have something to interface our webservice with. We decided to split off and go into two routes, a web app and an android app. They both have the same functionality but we wanted to create an android app specifically so that we can eventually port it over to an Android powered virtual reality device, such as our original plans with the Google Glass or Vuzix m100. We finished both of these apps and that was our final product for the semester.

Parts Used

- Microsoft HoloLens
- Unity
- Google Cloud Vision
- Android phone
- Computer able to run Visual Studio 2015 with C# v5

Project Retrospective

Although we were not able to have a full implementation of our design, we did manage to prove our system is functional. Our web service functioned exactly as designed, and our Android application is fully functional. Looking back at our process, we should've allocated more of our time to integration with the AR devices. Although

there are benefits to making an Android application, we wanted to implement our initial idea of having an immersive environment.

There is some future work left on this project. The obvious would be as previously mentioned, integrate with an AR device. But with the framework we already have built, we can increase the functionality and scope of our product dramatically. We have the ability right now to recognize characters both handwritten and printed with a very promising efficiency. If we pair our service with some different available API's we can develop some impressive technologies. The first would obviously be importing the Google Translate API and the option to translate text into any language. This would theoretically allow people who don't speak a language to communicate with each other with just a pen, paper, and any device that is running our application. Also, since have the ability to make a digital version of any handwritten text, we could theoretically create a smart document scanner. Anyone wearing a device with our application would be able to create a digital copy of any text they encounter.

All in all, this project was interesting experience. It was challenging, and forced all of us to be cohesive and creative with our solutions. Although we did not technically finish everything we wanted, we managed to create a very powerful back end that, if implemented correctly, could have many different uses. Time was the major limiting factor in our project. But in just a semester, we created a product that everyone in our group can be proud of.