L’Addition: Splitting the Check, Made Easy

Alvin Heng
Department of Electrical Engineering
Stanford University
Stanford, CA
vinster@stanford.edu

Abstract—In this paper, I describe the workings of an iPhone application designed to quickly and easily split a restaurant bill amongst a group of people. With the application, a user can take a snapshot of a restaurant receipt and instantly figure out the individual totals that everyone has to pay through a simple and easy to use interface. The application uses the Tesseract OCR engine to extract words from the receipt, then performs text-processing to define individual items on the receipt. The process of figuring out what one has to pay from a group check can be reduced down to a few short minutes as a result of this application.

Keywords—OCR; iPhone Application; Splitting Checks; Receipts; Restaurants

I. INTRODUCTION

Coordinating dinner plans at a restaurant for a large party comes with many nuisances: picking a restaurant that will please everyone, making reservations far enough in advance to get a large enough table, and hoping your server does not mess up everyone’s order. However, perhaps the most time consuming part of it all is figuring out how to split the bill at the end. Someone will usually take out a calculator or begin writing a lot to add up everyone’s individual order. Sometimes people do the addition themselves and forget to consider tax and tip and end up underpaying. Some people complain that tax and tip shouldn’t be split evenly if they didn’t order as much as another person. Almost all of the time, everyone’s contributions somehow don’t add up to the total required to pay the bill. The goal of my application is to resolve this issue and make splitting the check a simple task.

II. INSPIRATION

Part of the inspiration for this application comes from a restaurant in Cerritos, California called “STACKED,” that is quite ahead of the times. At STACKED, there are iPads at every table from which customers can send in their food orders directly to the kitchen without having to wait around for service from waiters or waitresses. Since the food orders are managed by the iPad application, the bill is also handled using the iPad. There is a really nice interface that lets users drag and drop ordered items into different buckets so that individuals of a group can each see the total they are required to pay, as well as add in the appropriate amount of tip. This removes away all the fuss about properly splitting a check since it requires no writing, no calculations, and no estimates. However, this process works so well only because the restaurant supplies iPads at every table. It might be a while before every restaurant that serves large groups begins converting to a completely digital ordering/paying experience, so in the mean time, there is still a need for an application that can ameliorate this process. That application is L’Addition.

III. USAGE

The application flow of L’Addition is as follows: a user takes a picture of their receipt and within seconds is presented with a list of all the items on the receipt which are all grouped as “unpaid.” A user can then select one or more items on the list and proceed to “checkout.” This will take them to a page showing a list of their selected items, the subtotal, a tax and tip field, and the total amount due. The user can choose to enter a tip percentage, which will automatically get calculated into the total. Before the user is done, they must enter their name so that the application can keep track of the totals that everyone must pay. Once they are done, the application goes back to the page of listed items. All of the previously selected items are removed from the “unpaid” list and in the “paid” list is a list of all people who have already calculated their required total contributions. This process is repeated for every customer in the group until everyone has their total amount.

IV. ALGORITHM PIPELINE

The application pipeline has 4 major components: user input (photo of receipt), optical character recognition (OCR) on receipt items, text processing to parse receipt items (name, quantity and price) into objects, and presenting of data to users.

A. User Input

The user input phase is quite straightforward; a user takes a snapshot of the receipt and then is allowed to manually crop out the region of the receipt that contains the items and the prices. The necessity for cropping will be explained later in the Limitations section of this document. This cropped image is now ready for the OCR phase.
The application leverages Tesseract, an open source OCR engine initially developed at HP Labs and currently managed by Google. Tesseract performs a few preprocessing steps before the main OCR phase. First, it applies Adaptive Thresholding to the image, which results in a binary image of white text over a black background that simplifies the rest of the steps. Next, connected component analysis is performed to extract character outlines. These outlines are then assembled into Blobs, which are filtered out based on the median height and then organized into lines of text. This line finding phase actually handles skewed images by processing the Blobs by their x-coordinate while keeping track of the slope across the page, so there is no need to compromise image quality by having to perform a deskew phase. The lines of text are broken up into words depending on the spacing between each character. Finally, two passes are made to recognize words that weren’t satisfactory the first time by using the training data from the first pass. Finally the output is given as a string of words [2][4]. The Tesseract algorithm is outlined in Figure 1.

Interfacing Tesseract with the iOS SDK was not too difficult. The application simply allocates an instance of the Tesseract API and initializes it with a file containing training data for English words characters. Next, setImage, recognize, and getUTF8Text are called on the Tesseract instance to complete the OCR phase. The result is a string of singly spaced apart words found in the image.

C. Text Processing

Given a string of words, the application now has to parse and make meaning of the text on the receipt. Since the words correspond to only the itemized part of the receipt, a simple grammar is used to construct receipt item objects. A receipt item is defined like so:

\[
\text{Item : [quantity] name price}
\]

Here, there is another name parameter that may describe an add-on to the item, but doesn’t actually cost anything since it refers to the same item. Currently, L’Addition does not handle this properly, since the output of the Tesseract API only gives a string of singly spaced words, disregarding any information about lines. As a result, the sub name will get added as a menu item, which is still okay since users only have to worry about selecting what they ordered, and any extraneous items can be safely ignored. However, to build a truly robust system, preprocessing can be done on the receipt to break up the receipt image into several components where each component represents the entire content of one item and may be multiple lines, which can then be passed onto the OCR engine and processed individually. In any case, each item is parsed from the string of text and stored as an Item object. Once all items have been parsed, the list of items is displayed to the user.

D. User-Interface

Once the internal representation of receipt items has been constructed, all that is left is to provide the user with an interface to split the items amongst different customers in the group. This phase has no algorithmic sophistication; it simply lets user select items from a list, computes the subtotal, tax, optional tip, and total price of that selected list of items, and then stores an object containing the user’s name and the total price they are responsible for paying. Each user repeats this process for themselves and in the end, everyone has the exact amount they are required to pay without having to perform any manual calculations. This phase can be seen in Figure 2.
As is, the application works quite successfully. The biggest drawback is that the cropping feature that is currently implemented is a major hack, and requires a little bit of knowledge about how it’s implemented to know how to use it correctly. Nonetheless, if the image is cropped properly and of good enough quality, then the rest of the application process is quite streamlined. Fortunately, the Tesseract API also provides a method for querying the mean confidence value of the procedure. If it is below a certain threshold, the application will reprompt the user to take a new photo. As far as the practical applications of L’addition goes, not much preprocessing is required to take place on the images. One valid assumption is that the receipt given by the restaurant will not have suffered from being wrinkled and etched away from being inside of someones' pocket. Thus, as long as the user takes a well-lit photo, the OCR engine has a high chance of succeeding.

Despite the success of this application on several test cases, there are still limitations that exist. For one, the cropping phase that the user has to perform is a bit cumbersome. The reason why it is implemented like that is because there are so many different formats of receipts. Not every receipt has a clear divider between the region containing the items on the receipt and all other extraneous regions of the receipt. (See Figure 3.)

Thus, to simplify this ambiguity, the user is forced to manually crop the image. Another limitation is that receipts can get awfully long, especially when there are a lot of people in the group. Taking a photo that fits every item on the receipt with clear enough resolution for the OCR engine may not be possible. The current workaround is to take multiple images, and dealing with one portion of the items at a time.

VI. Future Work

The problem I attempted to solve was easing the process of splitting a restaurant check amongst group of people. Honestly, I think long-term solution still lies in the future and will resemble the approach that the restaurant STACKED took with completely digital orders/receipts. Nonetheless, I think that reality is not in the immediate future, and believe that this application is a good placeholder for now. Seeing as this working procedure was developed in the course of a 1-2 week class project, I think that making it robust enough to be released on Apple’s App-Store can be done very soon. One thing that I may want to look into is the use of another OCR engine. ABBYY FineReader OCR engine has been said to work better than Tesseract in many cases, although it is not free [3].

VII. Appendix A: Breakdown of Work

For this project, I worked alone. In retrospect, having a partner or two could have eased the work needed be done immensely. This project required me to learn how to use the Tesseract API, perform text processing on its output, and develop a smooth flowing application to tie everything together. Time I had to spend on developing the application took away from some of the image processing algorithmic sophistication I could have added to the project, but I think it was fun nonetheless to be able to work on each component of this project.

REFERENCES


