Highlights

Development of a Finger-Key Identification Module for a Touch Typing Trainer

Oscar Vian L. Valles, Dhong Fhel K. Gom-os

- Automated identification of what finger was used to press a key with an accuracy of 99.58% in a dataset of 942 keypresses.
- Finger-key identifications take an average of 0.083 seconds. This allows for finger-key identification for typing speeds up to 143.040 Words per Minute
- Two metrics for correct finger placement were created: Finger Placement Accuracy and Historical Finger Placement Accuracy

Development of a Finger-Key Identification Module for a Touch Typing Trainer

Oscar Vian L. Valles[a], Dhong Fhel K. Gom-os[a,*]

[a] *Center for Research in Intelligent Systems, Department of Computer Science, University of the Philippines Cebu, Philippines*

Abstract

Typing tests list out words for the user to repeat. These tests measure metrics that quantify the user's ability to type. However, conventional typing metrics do not measure correct finger placement. This aspect of typing may affect the user's health. It is also a crucial aspect of keyboard typing education. As such, a method to identify which finger is used to press which key is beneficial. This paper introduces a technique to achieve this by developing a finger-key identification module that utilizes computer vision algorithms to detect the keys in a keyboard, and a ready-made machine learning solution to track fingers while typing. This module was successful in finger-key identification with an accuracy of 99.58% in a dataset of 942 keypresses. The module also had an average finger-key identification time of 0.083 seconds, which allows the module to perform real-time finger-key identification for typing speeds up to 143.040 Words per Minute which is well beyond the median typing speed of the population. Furthermore, two metrics for correct finger placement were defined and were implemented in a proof-of-concept trainer: (1) Finger Placement Accuracy which computes the number of keys pressed using the correct finger in a typing test sequence, and (2) Historical Finger Placement Accuracy which evaluates the user's ability to press a certain key with the correct finger over multiple typing test sequences.

Keywords: touch typing, keyboard typing metrics, touch typing trainer, computer vision, finger tracking

*Corresponding author at: Cebu City, Cebu, Philippines

Email addresses: oscarvianvalles@gmail.com (Oscar Vian L. Valles), dkgomos@up.edu.ph (Dhong Fhel K. Gom-os)

1. Introduction

There are a lot of educational typing tests available that help people learn touch typing, including Monkeytype, TypeRacer, and Keybr. These typing tests list out words that are then typed out. The entered keys are then compared to check if the user has typed the expected letter (Bartnik, 2021). At the end of the test, the time taken is calculated, and certain metrics are given. Examples of metrics include Words per Minute (WPM) and Accuracy (Arif and Stuerzlinger, 2009). However, this method of examination leaves out a crucial part of typing — correct finger placement.

There have been studies that show the effect of keyboard typing on the human body. These studies have shown that keyboard typing affects our neck, shoulder, upper limb, wrist, arms, and fingers (Szeto et al., 2005; Baker et al., 2007b). In addition, it has been shown that 22% of computer users sustain musculoskeletal disorders of the upper extremity. This includes the neck, shoulder, hands, and wrists. (Gerr et al., 2002)

Incorrect placement of the fingers exacerbates these issues. These erroneous finger placements may cause the following hand and wrist positions: ulnar deviation, forearm pronation, and wrist extension (Serina et al., 1999). These three are hand and wrist positions that are common in all activities, however, prolonged periods in these positions, such as in typing, may cause injuries such as Carpal Tunnel Syndrom (Toosi et al., 2015) and musculoskeletal disorders of the upper extremity (Marklin et al. (1999) as cited in Baker et al. (2007a)).

These issues can be prevented through keyboard typing education. Keyboard typing has been a part of the curriculum for a long time (Hoot, 1986) and studies have continued to this day to continue to optimize and improve methods of teaching keyboard typing to students. By starting to teach touch typing to students early, these students will develop the potential for higher-level keyboard typing (Donica et al., 2018).

There have been tools created to facilitate the teaching of touch typing. Keyboarding without Tears is a web-based application and 36-week curriculum that teaches students touch typing by following the three stages of Motor Learning Theory (Learning Without Tears, 2020). Monkeytype and TypeRacer are two keyboard typing test websites where users type a predetermined phrase, quote, or random words, and metrics are given after the test (Bartnik, 2021; Epshteyn, 2021). Keybr is similar to Monkeytype and Typerace, in that they also have the users type a predetermined phrase, quote, or ran-

dom words. However, this application guides the user by using statistics to create typing lessons that are appropriate to the current typing proficiency of the learner (Keybr Development Team, 2021).

However, these typing tests only measure the ability to type the correct character and do not check correct finger placement. While educators may be able to weed out bad habits manually, it is impractical for them to check each student individually.

While there have been other studies that utilize hand and finger tracking as input methods. Examples include studies by Dorfmuller-Ulhaas and Schmalstieg (2001) which used finger tracking as an input method in augmented environments, Hsu et al. (2014) used a Kinect to track fingers to play virtual instruments, and Yousaf and Habib (2014) created a virtual keyboard that operates using finger tracking. However, these studies used hand and finger tracking as an alternative input method, rather than supplementing physical keyboard typing.

There is currently no solution to check for correct finger placement while typing on a physical keyboard. Thus, there is a need for automatically identifying which finger is used to press which key during typing — hereby defined as finger-key identification.

The paper aims to create a finger-key identification module that accurately identifies what finger was used to press a key at a point in time to help develop better typing habits and healthier typing ergonomics through the use of computer vision algorithms. The paper also aims to create metrics to quantify finger placement during the duration of keyboard typing.

2. Algorithm

2.1. Setup

The experimental setup and configuration controlled three key elements: the camera, the keyboard, and the environment. The camera used was a Logitech C920. It was placed directly above the keyboard and captured in 720p/30fps. The keyboard used was a light-colored 60% keyboard in a modified American National Standards Institute (ANSI) layout. This type of keyboard only has the alphanumeric portion of a full-size keyboard. The color of the surface the keyboard rested on was dark. The contrasting color of the surface and the keyboard was chosen to improve the performance of the algorithm in keyboard detection. The entire environment was evenly lit with a 6500k LED bulb rated at 9 watts outputting 700 lumens.

This setup was used to collect data points to measure the algorithm's ability to perform finger-key identification in a controlled environment.

2.2. Keyboard Detection and Key Mapping

A computer vision algorithm was created as a starting point for mapping the keys of the keyboard within a video. OpenCV (Bradski, 2000) was used as the image processing library for this step.

2.2.1. Get Image Map

The first portion of the algorithm creates an image map that is overlaid over the detected edges of the keyboard. This image map subdivides the keyboard into multiple Regions of Interest (ROIs) where each ROI represents a key.

To do so, the input frame from the camera was converted to an 8-bit grayscale image. The frame was then denoised by using a bilateral filter. This filter removes noise while maintaining the edges. A Sobel filter was then used for edge detection (Sobel, 2014). Thresholding was then performed using Otsu's algorithm (Otsu, 1979) to reduce the bit-depth of the image from 8 to 1. This reduction of bit depth improves the performance in finding contours. Contours were found using the algorithm of Suzuki and Abe (1985).

The detected contours were then simplified as the output of the algorithm results in multiple points. To find the four points of the edges of the keyboard, the contours were sorted and the largest one was selected. The Douglas-Peucker algorithm for line simplification Saalfeld (1999) was then performed to reduce the contour points to the minimum amount. The contour points were then sorted clockwise, as the algorithm of Suzuki and Abe (1985) does not guarantee the arrangement of the contour points. There is a need for the points to be ordered clockwise since each point will be paired to the edges of the image map, which were sorted in clockwise order.

The image map was then stretched over the keyboard by using a perspective warp, where the four points of the edges of the keyboard were the end position of the four points on the edges of the image map. Figure 1 presents a grayscale frame and the resulting image map from that frame.

In ideal cases, the number of points would be four, with each point corresponding to the edges of the keyboard. However, there were times when other objects would be within the frame, or they intersected with the keyboard. This would result in a contour that would be defined by more than four points — which caused the entire algorithm to fail.

(a) Input image that was converted to grayscale (b) Transformed image map

Figure 1: First and last step of getting the image map

2.2.2. Get Key Contour Points

The second portion of the algorithm pre-computes the contour points of each key in the keyboard based on the generated virtual map and the Key-Color Values map.

The color that corresponds to each key in the virtual map is stored in the Key-Color Value map. Each individual ROI was isolated using the color values stored in the map. The simplified contours of each ROI were then obtained using the same method in finding and simplifying the contour of the keyboard.

During testing using training data, a lot of the failures in finger-key identification were due to the tight fit of the contour to the key. The contour on its own did not give enough buffer for the placement of the finger. This buffer was needed as, in some instances, the finger used to press the key was not exactly on top of the key, but rather to its side. In addition, it was also observed that the finger tracking does not annotate the very edge of the fingertip, but rather, it labels somewhere in the middle of the fingernail. The algorithm accounts for this situation by adding a buffer. This was achieved by scaling the contour points by seven pixels on each side.

After the contours of all ROIs were obtained, a Key-Edge Coordinates map was generated.

2.3. Finger Detection and Tracking

MediaPipe Hands (Lugaresi et al., 2019) was used as the finger detection and tracking solution. This algorithm is composed of two ML models working

in conjunction to be able to detect the different parts of the hands and track them accurately. This algorithm outputs the positions of 21 hand and finger landmarks for each hand within the image.

2.4. Integration

The two algorithms were combined to accomplish finger-key identification The integration of the algorithm was a two-step process. The first step was to get the Key-Edge Coordinates map shown in Section 2.2. The second step runs whenever a key press has been detected. This step used the Key-Edge Coordinates map in conjunction with the finger tracking algorithm selected in Section 2.3.

The finger tracking data contained the pixel positions of each landmark of the hands within the frame. The edge coordinates of the ROI of the key pressed were obtained using the Key-Edge Coordinates map. Any fingertip landmarks that fit within the ROI were then returned.

3. Data Gathering

Ten sentences were gathered from novels and texts in the public domain. These sentences were used as typing test sequences. 3 sets of 10 videos were then captured with each video corresponding to one typing test sequence. Each set corresponds to three predetermined speeds of typing: slow (15wpm), average (35wpm), and fast (80wpm). These speeds were based on test data from Keybr Development Team (2021). Best effort was made to stick to the predetermined typing speeds, however, there were discrepancies and the exact speed was not maintained throughout the entire typing test sequence. During typing, errors were not consciously taken into account, and errors happened naturally as part of the typing process — with more errors happening more frequently as the typing speed increased.

Manual finger-key identification was performed for all key presses present in the video. The annotated data was then divided into training and test data at a ratio of 60:40, respectively.

3.1. Data Inventory

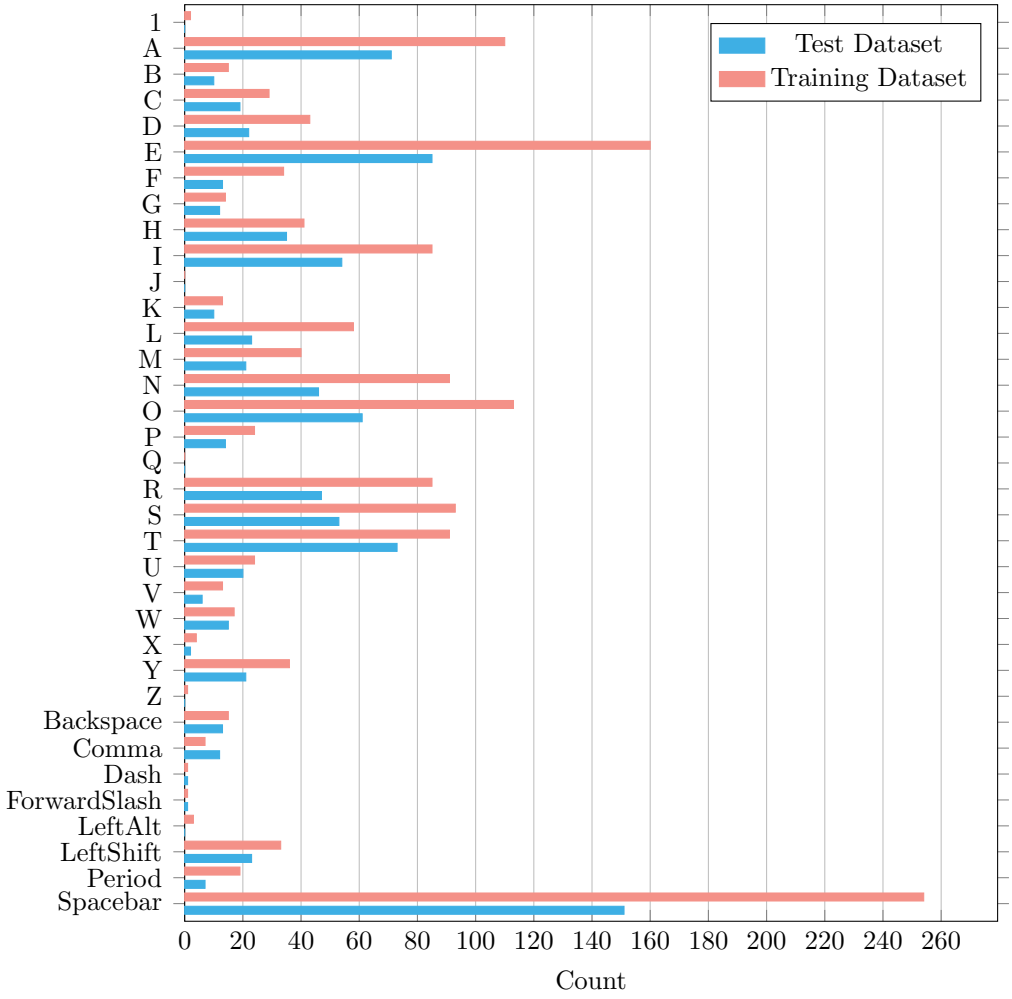

Figure 2: Data Inventory

Individual counts of the characters in the training and testing data set are shown in Figure 2. The letters were not represented equally in the data set. Of note, J and Q were not in the training dataset, and J, Q, and Z were not in the testing dataset. The inequality of representation in both of these datasets can be attributed to the source materials of the typing test sequences — novels and texts. There was no conscious decision to equally represent all characters in the data set.

4. Training and Testing

A script was created to test the accuracy of the module. This script opened each video and the associated labeled file. There were three frames taken for each video to perform step one of the module: frames ten, fifteen, and twenty. The first frame that returned a Key-Edge Coordinates map was used. The script then parsed the labeled file and opened the frame for each keypress. The second algorithm of the module was then run. The result of the module was then compared with the manual labeling of the keypress. Adjustments were then made to the algorithm after each training pass to improve its performance.

The same script was run through the test data.

5. Metrics

There were two total metrics obtained pertaining to finger and key mapping. The first was per test sequence, and the second was per key.

5.1. Per Test Sequence

The metric which calculates per test sequence is Finger Placement Accuracy (FP ACC). This metric computes the percentage of accurate keys pressed with the correct finger over the length of the typing test sequence. Inputting the wrong character, even if pressed with the correct finger, reduces FP ACC since FP ACC measures *accurate key presses*, and incorrect characters are considered inaccurate key presses.

The equation for calculating FP ACC is as follows:

$$FPACC = \frac{|F|}{|T|} \cdot 100\% \tag{1}$$

Where $|F|$ refers to the number of accurate keys pressed with the correct finger and $|T|$ refers to the length of the text.

5.2. Per Key

This metric, Historical Finger Placement Accuracy (HFP ACC), computes the accuracy of the user in pressing a certain key with the correct finger over all test sequences. The same criteria in determining accurate key presses. This allows the user to easily verify which keys need attention and training if there is a high frequency of error.

The equation for calculating a key's HFP ACC is as follows:

$$HFPACC_{char} = \frac{|F_{char}|}{|C_{char}|} \cdot 100\% \tag{2}$$

Where $|F_{char}|$ refers to the number of accurate keys pressed with the correct finger for a certain character, identified as $char$. $|C_{char}|$ refers to the number of times the character has appeared in all test sequences for the user.

6. Proof of Concept Trainer

A trainer was developed as a proof of concept of a real-world implementation of the finger-key identification module. It consisted of three screens. The first screen is where the keyboard was captured to perform get the Key-Edge Coordinates map. The second screen is where the typing test is performed and where real-time finger-key identification was done on each keypress. The third screen shows the computed metrics as defined in Section 5. Of note, the metrics shown on this page that uses multiple test sequences as an input use tests that were obtained only during the current training session. This proof of concept does not store the data of a user's previous test sequences that were performed from a different session.

7. Finger-Key Identification Accuracy

Accuracy, in the context of the finger-key identification module, calculates the percentage of successful finger-key identifications that the module has performed. In this case, success is defined as having the expected finger, based on the manual labeling, equal to the return value of the module. Accuracy is calculated as:

$$ACC = \frac{|S|}{|T|} \tag{3}$$

Where $|S|$ is the number of successful finger-key identifications, and $|T|$ is the total number of keypresses for all typing test sequences. This is also referred to as the success percentage.

7.1. Training Iterations

There were a total of seven training iterations. These training iterations served as a way to improve the performance of the module by adjusting its parameters.

7.1.1. Identifications

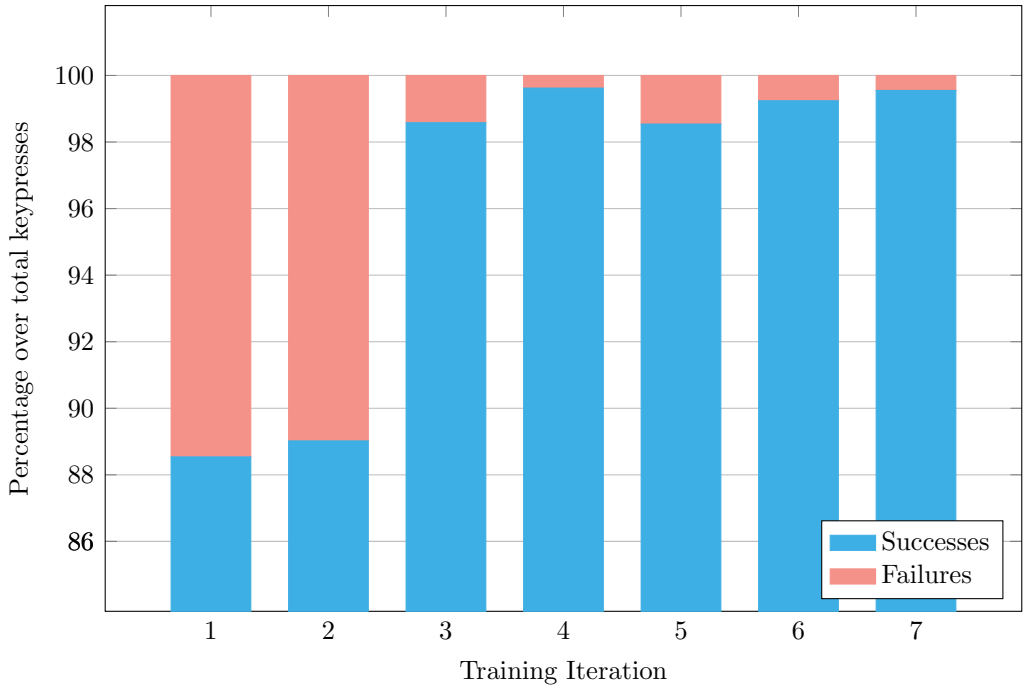

Figure 3: Identification results of the training iterations

Over the seven training iterations, there was an increase in success percentage as shown in Figure 3. Consequently, there was also a decrease in failure percentage. There was a total of 1475 total keypresses for iterations one to three, and 1571 total keypresses for iterations four to seven.

The increase in total keypresses starting from iteration four was due to adding multiple tries to the training routine in obtaining the Key-Edge Coordinates map. Iterations one through three only tried frame 10 as the source frame for the module while iterations four through seven tried frames 10, 15, and 20.

There was a need to try different frames since the camera was not consistent in its ability to focus on the keyboard for each video. In some videos video, the camera was not focused on the keyboard in frame 10, but it was able to focus on frame 15. As a result, the total number of available keypresses to test increased.

The 0.48% jump between iterations one and two can be attributed to cor-

recting erroneous finger-key identifications that were obtained from manual labeling. The 9.56% jump in success percentage between iterations two and three was due to adding buffers around the ROI by scaling the contours by 10 pixels on each side. Iteration four's increase was due to fixing the image map. Certain keys had incorrect color values assigned in the Key-Color Values map, and some keys were of incorrect shape. Iterations five to seven were performed to test different values and shapes for scaling the ROI.

7.1.2. Uncertain Successes

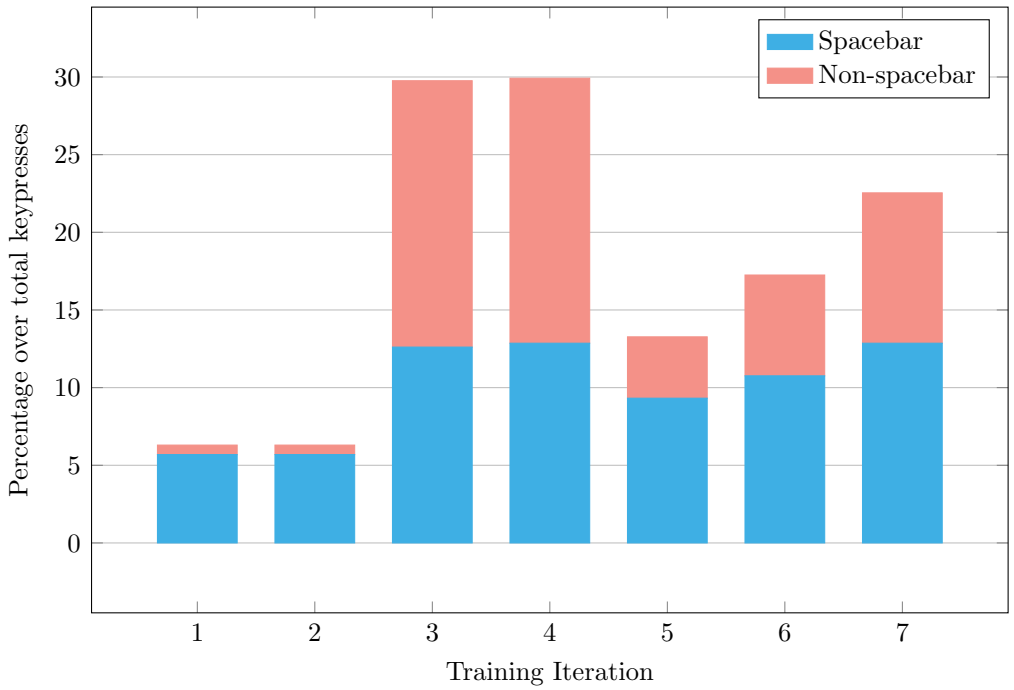

Figure 4: Uncertain Successes of the training iterations

Uncertain Successes are instances where the finger-key identification module was able to detect two or more fingertips within the ROI of the key. As such, the module is uncertain which of the multiple fingertips pressed the key. However, it is still classified as a success since the expected finger from the manual labeling exists within the array of fingertips returned by the module. Figure 4 shows the trend of this metric throughout the different training iterations.

Spacebar. In a 60% keyboard, the spacebar is the longest key on the keyboard. In addition, this key is right below the non-dominant hand's thumb's default resting position when the hand's posture follows the proper touch typing posture. As such, it was to be expected that a majority of the uncertain successes were because of the spacebar. However, this was not the case for iterations three and four. This was because the buffer for the ROI was too big and this caused the module to detect more than one fingertip within that ROI.

Non-spacebar. There were cases where the module had uncertain successes that were not over the spacebar. In almost practically all cases, these were over the alphanumeric keys. These uncertain successes stem from the increased buffer of the ROI. This type of uncertain success carries more weight compared to spacebar uncertain successes since the keys where these uncertain successes come from are the smallest keys on the keyboard. This means that the chances that a person has two or more fingers over the keys are rare, compared to the spacebar.

7.1.3. Uncertain Successes vs Success Percentage

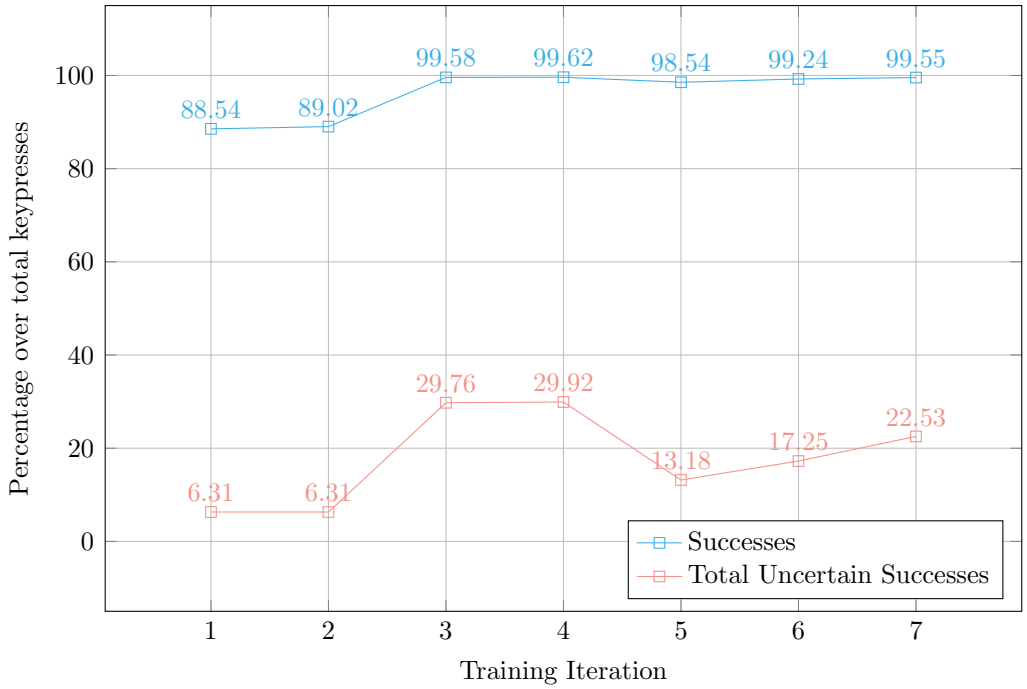

Figure 5: Uncertain Successes of the training iterations

Tuning the algorithm required managing both uncertain successes and the success percentage and making compromises between one or the other. The goal of the training process was to maximize the success percentage while limiting the number of uncertain successes.

As seen in Figure 5, it can be noted that in iterations one and two, the success rate was lower compared to the other iterations. However, the percentage of uncertain successes was also the lowest. Iteration three was the first iteration that created a buffer around each key, however, this buffer of 10 pixels was too aggressive, with the uncertain successes spiking upwards to nearly 30%. Iteration five reduced the buffer size by half to five pixels. This did not affect the success rate that much, with only a reduction of 1.08%. This also greatly decreased the uncertain success percentage by 16.74%. The sixth iteration met halfway between iterations four and five, by adding a buffer of seven pixels. This increased the success rate back to 99.24% without greatly increasing the number of uncertain successes. In comparison with the

fourth iteration, the sixth iteration had its success rate decrease by 0.38%, but its percentage of uncertain successes decreased by 12.67%. A seventh iteration was trialed where the increase in buffer was asymmetrical. For each key, the points nearer the user were increased by 10 pixels, while the points farther away were increased by five pixels only. This slightly increased the success rate by 0.31%. However, the percentage of uncertain successes also increased by 5.28%.

While the percentage of uncertain successes in iterations five through seven was in the range of 13% to 20%, The majority of these uncertain successes were spacebar uncertain successes, as shown in Figure 4.

Based on these results, the final selected parameters for the module were the parameters set in iteration six, with the buffer set to seven pixels as it was a good middle ground between success rate and uncertain successes.

7.1.4. Failure Types

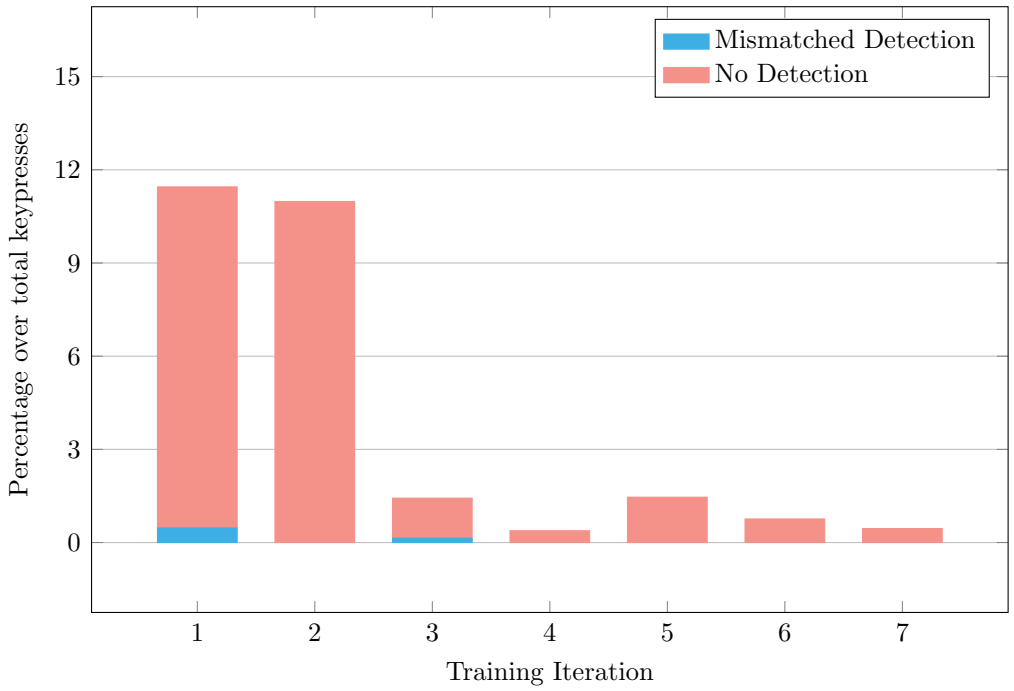

Figure 6: Uncertain Successes of the training iterations

Mismatched Identification. These are failures where the module did not return the expected fingertip based on the manual labeling. This type of failure

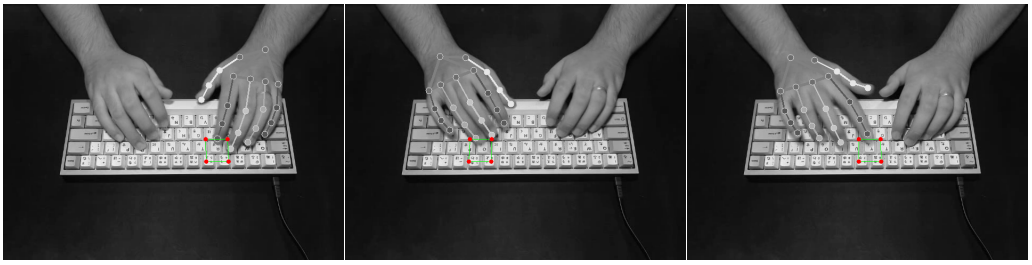

Figure 7: Example of No Identification Failures

only manifested in iterations one and three. The failures in iteration one, after double-checking the video, were due to an error during manual labeling. In iteration three, these errors were due to an incorrect image map and Key-Color Values map. The information for some keys was not consistent between both.

Based on this information and from the gathered data as shown in Figure 6, this failure type is non-existent for the module in real-life conditions.

No Identification. These are failures where the module did not detect any fingertips within the ROI. Figure 7 shows examples of frames where these errors occurred. These made up all the failures from the test results. These types of failures occurred because the fingertips used to press the key were not directly above the key, but rather offset to its side. Scaling the ROI would reduce the number of no identification failures, but it would also increase the number of uncertain successes. See Section 7.1.3 for more information.

7.1.5. Occlusion

The module was also successful in finger-key identification, even if both the key and finger were occluded. Figure 8 is one frame where this occurred. The module was successful in identifying that the Left Thumb was used to press the Left Shift.

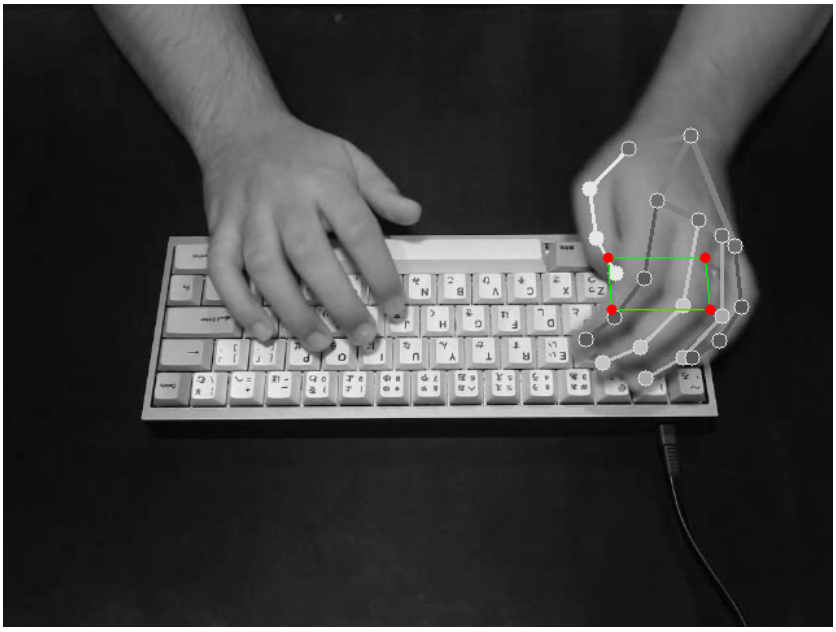

Figure 8: Occluded finger and key

7.2. Test Results

Table 1: Accuracy Results of the Module with Test Data

Category	Value
Total Keypresses	942
Identifications	
Successes	938
Failures	4
Success Percentage / Accuracy	99.58%
Failure Percentage	0.42%
Failure Types	
Mismatched Identification	0
No Identification	4
Uncertain Successes	
Total	133
Spacebar	72
Spacebar Percentage	54.14%
Spacebar Percentage (Total)	7.64%
Non-spacebar	61
Non-spacebar Percentage	45.86%
Non-spacebar Percentage (Total)	6.48%

Table 1 provides the accuracy results of the module with test data. These results were satisfactory after running the module with the parameters from the sixth training iteration on the test data. There was another run with the same test data and the same parameters, however, these results were omitted since this run had inconclusive results as there was a single identification that was incorrectly labeled during manual finger-key identification.

8. Finger-Key Identification Speed

8.1. Training Iterations

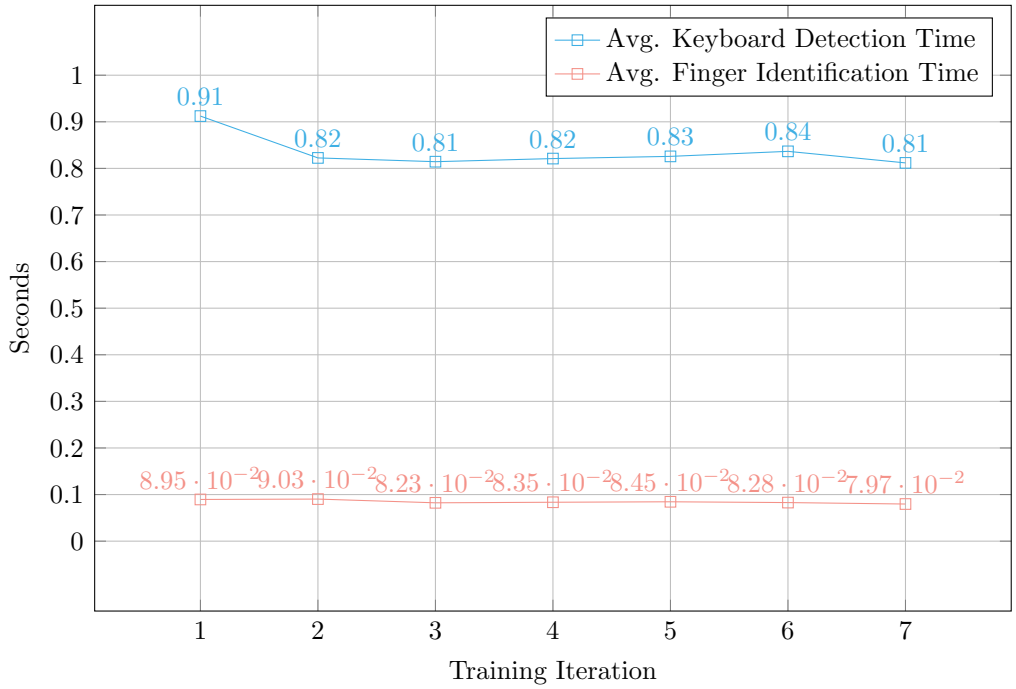

Figure 9: Keyboard detection time of the training iterations

The training iterations had an average keyboard detection time of $0.8620 \pm 0.0502s$ and an average finger identification time of $0.0863 \pm 0.0040s$. Throughout the training, there was no noticeable difference in running time over the same dataset as shown in Figure 9. However, the first training iteration had issues with gathering the speed of the module, as the average keyboard detection time included other operations that were not part of getting the Key-Edge Coordinates map. Removing this iteration, the average keyboard detection time becomes more consistent, with it being $0.8295 \pm 0.0070s$.

8.2. Average Keyboard Detection Time

This is the time that was measured as the module generated the Key-Edge Coordinates map. This corresponds with the first step of the module. It is to be noted that this time does not affect keyboard typing and is not

limited by the 0.32 threshold as the user is not yet typing at this point. The module got an average keyboard detection time of 0.083s when ran on the test data.

8.3. Average Finger Identification Time

This is the time spent in finger-key identification for a single key. As such, this time should be below 0.32 seconds for the module to be capable of performing finger-key identification in real-time without noticeable slowdown from the perspective of the user.

The average finger identification time that the module got when ran on the test data was 0.083s which is well below 0.32s. This can be attributed to the speed of the MediaPipe library, and the pre-calculation of the edge coordinates.

8.3.1. Real-Time Finger-Key Identification

Based on the definition of WPM from Arif and Stuerzlinger (2009), it can be extrapolated that calculating Characters per Second (CPS) is:

$$CPS = \frac{WPM}{60} \cdot 5 \tag{4}$$

The average WPM when typing using the QWERTY layout, according to test data from Keybr Development Team (2021) is ≈ 37.5 WPM. As such, we can calculate that the average Characters per Second when typing using the QWERTY layout is 3.125. We can calculate the seconds needed per character by this equation:

$$s = \frac{1}{CPS} \tag{5}$$

This means that each character, on average, requires 0.32 seconds to type. This sets the maximum time spent for finger-key identification for a single character. Any more than this would result in a noticeable slowdown during typing.

With each keypress only requiring 0.083s for finger-key identification, it can be gathered that the maximum CPS which the module can still perform real-time finger-key identification is 11.9200 CPS, based on Equation 5. It follows that the maximum WPM that still allows for real-time finger-key identification is 143.040 WPM, based on Equation 4. This is well above the median of 37.5 WPM on Keybr. The module can also perform real-time

finger-key identification for 90.99% of the community members of Monkeytype with the speed role.

9. Conclusion

The finger key identification module was successfully developed and implemented in this study.

The finger-key identification solution was split into two parts, keyboard detection and key mapping, and finger detection. Keyboard detection and key mapping used the following algorithm and techniques: (1) Edge Detection using Sobel filter (Sobel, 2014), (2) Thresholding using Otsu's algorithm (Otsu, 1979), (3) Finding contours using the algorithm of Suzuki and Abe (1985), (4) line simplification using the Douglas-Peucker algorithm (Saalfeld, 1999), and (5) Perspective transform (OpenCV, 2022). Finger detection utilized a ready-made solution called MediaPipe Hands by Lugaresi et al. (2019). The two were combined to perform finger-key identification.

The keyboard used was a 60% keyboard in ANSI. It was light in color and was connected with a black USB-C cable. The surface of the desk was dark, and a LED Bulb rated at 9 Watts, 700 lumens, and 6500k color temperature was used to uniformly light the capture area. The single optical camera, a Logitech C920, was placed above the keyboard, pointing directly downwards. It captured videos in 720p/30fps with a diagonal field of view of 78°. The computer used had an AMD Ryzen 5 3600 CPU, AMD Radeon RX 5600 XT GPU, G.Skill Trident Z Neo RGB 16GB 3200mhz RAM, and Samsung SSD 850 EVO SSD.

The development of the module utilized OpenCV as the main image manipulation platform that implemented the algorithms mentioned previously. OpenCV was also used to capture videos from the optical camera. It was all written in Python.

The module was accurate in 99.58% of identifications in a data set composed of 942 keypresses. However, 14.12% of the identifications were uncertain successes. These are successes where more than one fingertip was found in an ROI. Of these, 6.48% were non-spacebar uncertain successes. These are uncertain successes over keys that are not the spacebar. This bears more weight since most of these keys are the smallest on the keyboard, and in most cases, only one fingertip could fit within the key. These uncertain successes were due to the addition of a buffer around each ROI to increase the success rate.

The module is fast enough for real-time finger-key identification up to typing speeds of 143.040 WPM. This is above the median typing speed of 37.5 WPM on Keybr (Keybr Development Team, 2021). This also handles 90.99% of the 12678 members with the speed role on Monkeytype.

Two metrics were developed and implemented in a proof of concept trainer: (1) Finger Placement Accuracy which computes the percentage of keys pressed with the correct figure over the length of the test sequence, and (2) Historical Finger Placement Accuracy which computes the number of times the user is successful if pressing a key with the correct finger over all test sequences.

10. Future Work

There are a couple of ways to improve upon the module and its application. (1) While a trainer was developed, this trainer is lacking in functionality to be a usable trainer for teaching touch typing. (2) There were no tests done to prove the effectiveness of the module on a learner's skill in touch typing, and there was also no test done to prove that training using the module will improve posture and reduce problematic hand positions and movements. (3) The environmental setup was tightly controlled. This is not representative of real-world conditions. (4) The keyboard used was specific. Supporting other keyboard types, layouts, and color schemes would greatly increase the number of users that could use the module. (5) The buffer of the ROI was selected through trial and error. Adaptive techniques or machine learning algorithms could select better values for the buffer. (6) The module is limited in its ability to track keyboards that move between keypresses. (7) The dataset used did not have an equal representation of all characters within the keyboard and could be improved.

11. CRediT authorship contribution statement

Oscar Vian L. Valles: Conceptualization, Methodology, Software, Investigation, Resources, Data Curation, Writing — original draft, Writing — review & editing, Visualization, Funding acquisition. **Dhong Fhel K. Gom-os**: Conceptualization, Methodology, Writing — original draft, Writing — review & editing, Supervision, Project administration

12. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

13. Acknowledgment

We would like to recognize that this research was partially funded by the Department of Science and Technology — Science Education Institute

References

Arif, A., Stuerzlinger, W., 2009. Analysis of text entry performance metrics, pp. 100–105. doi:10.1109/TIC-STH.2009.5444533.

Baker, N., Cham, R., Cidboy, E., Cook, J., Redfern, M., 2007a. Kinematics of the fingers and hands during computer keyboard use. Clinical Biomechanics 22, 34–43. doi:10.1016/j.clinbiomech.2006.08.008.

Baker, N., Cham, R., Hale, E., Cook, J., Redfern, M., 2007b. Digit kinematics during typing with standard and ergonomic keyboard configurations. International Journal of Industrial Ergonomics 37, 345–355. doi:10.1016/j.ergon.2006.12.004.

Bartnik, J., 2021. URL: https://monkeytype.com/about. publication Title: Monkeytype.

Bradski, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools .

Donica, D.K., Giroux, P., Faust, A., 2018. Keyboarding instruction: Comparison of techniques for improved keyboarding skills in elementary students. Journal of Occupational Therapy, Schools, & Early Intervention 11, 396–410. doi:10.1080/19411243.2018.1512067.

Dorfmuller-Ulhaas, K., Schmalstieg, D., 2001. Finger tracking for interaction in augmented environments, in: Proceedings IEEE and ACM International Symposium on Augmented Reality, pp. 55–64. doi:10.1109/ISAR.2001.970515.

Epshteyn, A., 2021. About TypeRacer. URL: https://data.typeracer.com/misc/about.

Gerr, F., Marcus, M., Ensor, C., Kleinbaum, D., Cohen, S., Edwards, A., Gentry, E., Ortiz, D., Monteilh, C., 2002. A prospective study of computer users: I. Study design and incidence of musculoskeletal symptoms and disorders. American Journal of Industrial Medicine 41, 221–235. doi:10.1002/ajim.10066.

Hoot, J.L., 1986. Keyboarding Instruction in the Early Grades: Must or Mistake? Childhood Education 63, 95–101. URL: https://doi.org/10.1080/00094056.1986.10521749, doi:10.1080/00094056.1986.10521749. publisher: Routledge _eprint: https://doi.org/10.1080/00094056.1986.10521749.

Hsu, M.H., Shih, T.K., Chiang, J.S., 2014. Real-Time Finger Tracking for Virtual Instruments, in: 2014 7th International Conference on Ubi-Media Computing and Workshops, pp. 133–138. doi:10.1109/U-MEDIA.2014.53.

Keybr Development Team, 2021. keybr.com - Typing lessons. URL: http://www.keybr.com/.

Learning Without Tears, 2020. Keyboarding Without Tears | K-5. URL: https://issuu.com/handwritingwithouttears/docs/kwtbrochure2020/1.

Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.L., Yong, M., Lee, J., Chang, W.T., Hua, W., Georg, M., Grundmann, M., 2019. MediaPipe: A Framework for Perceiving and Processing Reality, in: Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR) 2019. URL: https://mixedreality.cs.cornell.edu/s/NewTitle_May1_MediaPipe_CVPR_CV4ARVR_Workshop_2019.pdf.

Marklin, R.W., Simoneau, G.G., Monroe, J.F., 1999. Wrist and forearm posture from typing on split and vertically inclined computer keyboards. Human Factors 41, 559–569. doi:10.1518/001872099779656770.

OpenCV, 2022. The OpenCV Reference Manual. 4.6.0 ed., OpenCV. URL: http://docs.opencv.org/.

Otsu, N., 1979. A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man, and Cybernetics 9, 62–66. doi:10.1109/TSMC.1979.4310076. conference Name: IEEE Transactions on Systems, Man, and Cybernetics.

Saalfeld, A., 1999. Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. Cartography and Geographic Information Science 26, 7–18. URL: https://doi.org/10.1559/152304099782424901, doi:10.1559/152304099782424901. publisher: Taylor & Francis _eprint: https://doi.org/10.1559/152304099782424901.

Serina, E.R., Tal, R., Rempel, D., 1999. Wrist and forearm postures and motions during typing. Ergonomics 42, 938–951. doi:10.1080/001401399185225.

Sobel, I., 2014. An Isotropic 3x3 Image Gradient Operator. Presentation at Stanford A.I. Project 1968 .

Suzuki, S., Abe, K., 1985. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing 30, 32–46. URL: https://www.sciencedirect.com/science/article/pii/0734189X85900167, doi:10.1016/0734-189X(85)90016-7.

Szeto, G., Straker, L., O'Sullivan, P., 2005. A comparison of symptomatic and asymptomatic office workers performing monotonous keyboard work - 2: Neck and shoulder kinematics. Manual Therapy 10, 281–291. doi:10.1016/j.math.2005.01.005.

Toosi, K.K., Hogaboom, N.S., Oyster, M.L., Boninger, M.L., 2015. Computer keyboarding biomechanics and acute changes in median nerve indicative of carpal tunnel syndrome. Clinical Biomechanics 30, 546–550. doi:10.1016/j.clinbiomech.2015.04.008.

Yousaf, M., Habib, H., 2014. Virtual Keyboard: Real-Time Finger Joints Tracking for Keystroke Detection and Recognition. Arabian Journal for Science and Engineering 39, 923–934. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84893135336&doi=10.1007%2fs13369-013-0909-2&partnerID=40&md5=1b6ad8dd4d2d32f8372fcdefb92966f1, doi:10.1007/s13369-013-0909-2.