

Copyright  
by  
Veneela Nagabandi  
2018

DESIGNING TO LEARN TANGIBLE PROGRAMMING

by

Veneela Nagabandi, B. Tech

THESIS

Presented to the Faculty of  
The University of Houston-Clear Lake  
In Partial Fulfillment  
Of the Requirements  
For the Degree

MASTER OF SCIENCE

in Software Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

MAY, 2018

DESIGNING TO LEARN TANGIBLE PROGRAMMING

by

Veneela Nagabandi

APPROVED BY

---

Soma Datta, Ph.D., Chair

---

Michael Findler, Ph.D., Committee Member

---

Khondker Hasan, Ph.D., Committee Member

APPROVED/RECEIVED BY THE COLLEGE OF UNIVERSITY OF  
HOUSTON- CLEAR LAKE:

---

Dr. Said Bettayeb, Ph.D., Associate Dean

---

Ju H. Kim, Ph.D., Dean

## **Dedication**

To

The Field of Software Engineering and My family

## **Acknowledgements**

I would like to express my sincere thanks, appreciation, and gratitude to all of those who have directly or indirectly contributed to my research work and those who have supported me throughout the entire process. I will always be grateful for that.

I would like to thank my supervisor Dr. Soma Datta for her excellent guidance and engagement through my thesis period. She has been supportive since the day I began working on my research. Her insightful discussions and suggestions on the research helped me finish this thesis successfully. Moreover, I would like to thank my thesis committee, Dr. Michael Findler, and Dr. Khondker Hasan, for their encouragement and their helpful advice.

My immense love goes to my grandparents, Tataya(Grand Father) Lakshmi Narayana Daram, Amma(Mummy) Rama Devi Nagabandi, Nana(Daddy) Ram Badraiah Nagabandi, (Mavaya) Jaya Prakash Daram. I cannot thank them enough for their unconditional love, support and care through all these years. I would not have made it this far without them. Loads of love and thanks to my brother's Karthik Nagabandi and Tarun Kumar Reddy Mudireddy, who blossomed and cherished with me every great moment and supported me by keeping me harmonious and helping me putting pieces together.

Finally, I would like to thank my friends, roommates, Sasank Ambadasugari for the User Interface. Vamshi Krishna Velishetti, Sai Kumar Reddy Sheru, Sai Kiran Ippili, Sai Charith Daggupati, Deepika Rapolu, Veena Namani, Sruthi Paritala, and Varsha Bondugula for tapping me all the time to get my thesis progressed and for a wonderful love they pour on me, making my life joyful.

ABSTRACT  
DESIGNING TO TANGIBLE PROGRAMMING

Veneela Nagabandi  
University of Houston-Clear Lake, 2018

Thesis Chair: Dr. Soma Datta

The thesis describes a ubiquitous technique for learning tangible coding in R programming language for middle and late elementary school students. It emphasizes the use of inexpensive and durable wooden blocks with no embedded power supplies. These blocks are shaped like the pieces of wooden cubes which contains basic syntax, functions, decision making. Students integrate these wooden blocks to create a R programming statement in offline settings such as on desks or floor in the classroom. An image of the tangible code is captured using phone and uploaded to ‘R’ programming language through command line. The image representation is finally executed using R interpreter. Alternatively, they can also learn programming through drag and drop Interface. This tangible programming technique stimulates interests among young students. It can help middle and high school students to develop analytical skills, logical thinking, and affection for coding. The hypothesis of this pedagogy is “Programming can be for all ages and be learnt by

themselves with minimal given tools”. Moreover, this learning approach at an early age helps remembering code syntax and offers more retention of programming syntax rate than traditional classroom intangible programming [1].

## TABLE OF CONTENTS

List of Tables .....	x
List of Figures .....	xi
Chapter .....	Page
CHAPTER I: INTRODUCTION.....	1
CHAPTER II: MOTIVATION AND CONTEXT.....	6
What is programming?.....	6
What is tangible programming?.....	6
Why tangible programming? .....	6
Collaborative Programming.....	7
Debugging.....	7
Limitations of “visual” programming languages.....	7
CHAPTER III: RELATED WORK.....	9
CHAPTER IV: METHODOLOGY .....	12
Capturing the images using Camera .....	13
Wooden blocks.....	13
Importing the images into R software.....	14
Applying image processing techniques on the captured images using imager package .....	15
Overview of imager package .....	15
Grayscale image.....	15
Thresholding .....	16
Pixel .....	16
Shrink and grow pixels .....	17
Extracting text from images using Optical Character Recognition .....	18
Evaluating the extracted text using R software.....	18
Automating the tangible programming using the User-interface .....	19
User interface .....	19
CHAPTER V: EXPERIMENTAL AND RESULTS.....	20
Minimum function .....	20
VI. CONCLUSION AND FUTUREWORK .....	31
Conclusion .....	31

Future Work .....	31
REFERENCES .....	33
APPENDIX A: OTHER EXPERIMENTAL RESULTS .....	35
Square root Function.....	35
Sum Function .....	39
Maximum Function.....	43
Log Function .....	47
Round Function.....	51
Mean Function .....	55

## LIST OF TABLES

Table	Page
Table 4.1 Syntax and cube color .....	13
Table 4.2 Threshold value for respective function image.....	16
Table 4.3 Shrink and grow values for respective function image.....	17

## LIST OF FIGURES

Figure	Page
Figure 4.1 Overall workflow.....	12
Figure 4.2 An example captured image .....	13
Figure 4.3 Importing an image into R software .....	14
Figure 4.4 Input and Output to R software .....	18
Figure 5.1 A colored image which contains minimum function .....	20
Figure 5.2 After converting the minimum colored image to greyscale image .....	20
Figure 5.3 After converting the greyscale image into data frame.....	21
Figure 5.4 After applying the linear model on the minimum function greyscale image data frame.....	21
Figure 5.5 Plot before and after trend removal from the greyscale minimum function image .....	22
Figure 5.6 Plot before and after shrink, then grow pixels of the minimum function greyscale image.....	22
Figure 5.7 Displaying the number of pixels in a minimum function greyscale image.....	23
Figure 5.8 Extracting the text from image and evaluation.....	23
Figure 5.9 User Interface of UHCL e-learning system.....	24
Figure 5.10 Calculating the sum of 6 numbers .....	25
Figure 5.11 Evaluating the print function .....	26
Figure 5.12 Passing the value to the variable and displaying.....	27
Figure 5.13 Evaluating the if statement and displaying.....	28
Figure 5.14 Evaluating the switch statement and displaying.....	29
Figure 5.15 Reset the user interface.....	30
Figure A.1 A colored image which contains square root function .....	35
Figure A.2 After converting the square root image into greyscale image .....	35
Figure A.3 After converting the greyscale image into data frame.....	36
Figure A.4 Applying linear model on the data frame of square root function greyscale image.....	36
Figure A.5 Before and after trend removal from square root function greyscale image.....	37

Figure A.6 Before and after shrink, then grow pixels of square root function greyscale image.....	37
Figure A.7 Displaying the number of pixels in a square root function grey scale image.....	38
Figure A.8 Extracting and evaluating text from square root function greyscale image.....	38
Figure A.9 A colored image which contains sum function .....	39
Figure A.10 After converting the sum function colored image into greyscale image.....	39
Figure A.11 After converting the sum function greyscale image into data frame.....	40
Figure A.12 Applying linear model on the data frame of sum function greyscale image.....	40
Figure A.13 Before and after trend removal from sum function greyscale image .....	41
Figure A.14 Before and after shrink, then grow pixels from sum function greyscale image.....	41
Figure A.15 Displaying the number of pixels in a sum function grey scale image .....	42
Figure A.16 Extracting and evaluating text from sum function greyscale image .....	42
Figure A.17 A colored image which contains maximum function.....	43
Figure A.18 After converting the maximum function image into greyscale image .....	43
Figure A.19 After converting the maximum function greyscale image into data frame .....	44
Figure A.20 Applying linear model on the data frame of maximum function greyscale image.....	44
Figure A.21 Before and after trend removal from maximum function greyscale image.....	45
Figure A.22 Before and after shrink, then grow pixels from maximum function greyscale image.....	45
Figure A.23 Displaying the number of pixels in a maximum function grey scale image.....	46
Figure A.24 Extracting and evaluating text from maximum function greyscale image.....	46
Figure A.25 A colored image which contains log function .....	47
Figure A.26 After converting the log function colored image into greyscale image.....	47
Figure A.27 After converting the log function greyscale image into data frame .....	48

Figure A.28 Applying linear model on the data frame of log function greyscale image.....	48
Figure A.29 Before and after trend removal from log function greyscale image.....	49
Figure A.30 Before and after shrink, then grow pixels from log function greyscale image.....	49
Figure A.31 Displaying the number of pixels in a log function greyscale image .....	50
Figure A.32 Extracting and evaluating from log function greyscale image .....	50
Figure A.33 A colored image which contains round function.....	51
Figure A.34 After converting the round function colored image into greyscale image.....	51
Figure A.35 After converting the round function greyscale image into data frame .....	52
Figure A.36 Applying linear model on the data frame of round function greyscale image.....	52
Figure A.37 Before and after trend removal from round function greyscale image.....	53
Figure A.38 Before and after shrink, then grow pixels from round function greyscale image.....	53
Figure A.39 Displaying the number of pixels in a round function grey scale image.....	54
Figure A.40 Extracting and evaluating text from round function greyscale image.....	54
Figure A.41 A colored image which contains mean function .....	55
Figure A.42 After converting the mean function colored image into greyscale image.....	55
Figure A.43 After converting the mean function greyscale image into data frame.....	56
Figure A.44 Applying linear model on the data frame of mean function greyscale image.....	56
Figure A.45 Before and after trend removal from mean function greyscale image .....	57
Figure A.46 Before and after shrink, then grow pixels from mean function greyscale image.....	57
Figure A.47 Displaying the number of pixels in a mean function grey scale image .....	58
Figure A.48 Extracting and evaluating text from mean function greyscale image .....	58

## CHAPTER I: INTRODUCTION

Programming provides a way to understand information and content in almost every aspect of daily life. The major difference between a traditional programming language and a tangible programming language is that programming is explained in a more theoretical manner in the former, while the latter explains programming in a more actionable or practical manner. This helps in the process of learning and understanding the concept of programming and helps students to retain these concepts in the later stages of their lives. Integrating and teaching computer programming into classroom curriculum at an early age with limited resources can be a massive task.

Learning a programming language for novice programmers contains several activities, e.g., learning the language features, program design, and program comprehension. Distinctive approach in textbooks and programming courses is to start with declarative knowledge about a programming language. Several common deficits in novices' understanding of definite programming language include variable initialization which seems to be more difficult to understand rather than updating or testing variables. Bugs with especially loops and conditionals are common and actions that take place "behind the scene", like updating loop variables in "for" loops, are difficult for novice programmers.

Students have often many misconceptions in their understanding or implementing of recursion. Expressions that are syntactically close to each other or mean different things in different contexts cause practical difficulties, e.g. "123" and 123, in programming language. It is very important to distinguish between programming knowledge and programming strategies. A student can learn to explain and understand a programming concept, e.g., what does a loop mean, but still fail to use it appropriately in a program. A few students may know the syntax and semantics of individual statements, but they do not

know how to combine these features into valid programs. Even when they know how to solve the problem by hand, they have trouble translating it into an equivalent computer program. It takes quite a long time to learn the relation between a program on the page and the mechanism it describes.

Students have difficulties in understanding that each instruction is executed in the state that has been created by the previous instructions. For example, tangible computer programming language is an example of this kind of approach and was commonly used in the 80's in introductory programming education. It is since, all the instructions which include creating programs to control a robot movement on a display, thus making the program state and transformations clearly visible. There is often little correspondence between the ability to write a program and the ability to read one. Programming courses should include them both. In addition, some basic test and debugging strategies should be taught.

Another issue that complicates the learning of programming is the distinction between the model of the program as it was intended, and the program as it is. There are often mistakes in the design and bugs in the code. Also in working life, programmers need to understand a program that is running in an unexpected way. This requires an ability to trace code and to build a mental model of the program to predict its behavior. This is one of the skills that could be developed by emphasizing program comprehension and debugging strategies in the programming courses. Many classrooms have only limited computers available for writing programs due to lack of funding. Furthermore, since the lack of funding limits the number of teachers a school can hire, the students cannot learn and understand the syntax by teaching programming through limited resources. This is evident in countries like India where the resources are limited. The students needed to be divided into small teams to teach programming in such an environment. Learning

programming syntax for novice students will be difficult and need teachers' help frequently.

Tangible programming is very similar to text-based and visual programming language. However, instead of using pictures and words on a computer screen, a tangible programming language uses physical objects such as wooden blocks, to represent various programming elements, commands, and flow-of-control structures. Students integrate these wooden blocks to form a physical construction that describes a computer syntax. By giving programming a physical form, we believe that tangible languages have the potential to ease the learning of complicated syntax, to improve the style and tone of student collaboration, and to make it easier for teachers to maintain a positive learning environment in the class. The programming languages explained using this approach at an early age offers more retention rate than traditional screen-based classroom teaching. However, Tangible User Interfaces (TUI) in other researches are not without drawbacks; the technology involved is often delicate, expensive, and nonstandard, causing significant problems in classroom settings where cost is always a factor and technology is not dependable. Thus, to better explore potential benefits of tangible programming, this study began with the development of tangible computer programming languages that are inexpensive, reliable, and practical for classroom use.

This study attempts to address most of the aforementioned problems using teaching programming in a tangible way. It employs in-class room kinetic based tangible technology to teach computer statement to novice students while helping them to overcome possible fears of learning programming.

This thesis describes a unique technique for designing, implementing and teaching R language in a tangible way for middle and late elementary school students. It emphasizes the use of inexpensive and durable wooden blocks with no embedded power supplies.

These wooden blocks which contain basic syntax, functions, decision making and many more. Students integrate these wooden cubes to create a computer statement in offline settings such as on desks or on a floor and use a portable scanning camera which can be attached to the laptop or desktop to capture the pictures. The captured images are imported into R console and image processing is performed using various image processing techniques available in R packages such as EBImage, Magick, Imager and other similar packages. The text is extracted from the processed images using Optical Character Recognition (OCR) which is available in “tesseract” package in R. The common string operations on the extracted text are performed by using “stringr” package in R programming language. Finally, evaluation of the extracted text is performed using the “eval” function in R programming language.

Alternatively, the students can also learn programming automatically through Interface. The Interface contains drag and drop elements. Whenever user drags and drops the elements and click the submit button the syntax is evaluated in R interpreter and results are displayed on the User Interface. There is a “clear” button to clear the User Interface. This helps novice programmers to learn and understand the syntax while overcoming the problems faced due to limited resources such as desktops, laptops, tablets or teachers and difficulties associated with learning programming.

It is important to note that tangible programming languages are not yet commercially available, and their use has been restricted almost entirely to laboratory and research settings. This study teaches R programming language which is a data science language; learning in this language could help people understand the syntax of other commercial programming languages [1]. Thus, the advantages outlined above are hypothetical. Indeed, one of the primary goals of this project is to better understand how

commercial programming languages are taught using tangible methods, which might affect student learning in classroom environments compared to more conventional learning.

## CHAPTER II: MOTIVATION AND CONTEXT

Here, the motivation and context of the thesis is explained using various programming contexts such as a tangible programming which explains the “tangible programming language” was initially coined by Suzuki and Kato [1] to describe their AlgoBlock, collaborative programming explains the teamwork, visual programming and the most problematic part of learning how to program i.e., Debugging.

### **What is programming?**

For the purposes of this thesis, programming is defined very broadly. Programming ranges from simple to complex. It nearly takes years, for a software engineers to write programs that varies from some hundreds to thousands of pages but programs written by children are immensely small and uncomplicated which produces very intriguing outcomes. The thesis primarily focuses on programs of this scale. Simple programs can be greatly empowering to children and adults alike.

### **What is tangible programming?**

The term “tangible programming language” was initially coined by Suzuki and Kato [1] to describe their AlgoBlock collaborative programming environment for kids. The unique feature that separates AlgoBlock apart from other programming environments is the understandable nature of its user interface. Instead of manipulating virtual objects displayed on a computer screen, users of AlgoBlock arranged physical blocks on a table to communicate to the computer. “Tangible programming” refers to the activity of arranging the blocks to build computer programs.

### **Why tangible programming?**

There are several reasons why one might want to program by manipulating physical objects. Some people, mostly children, learn more readily when physical objects are involved in the learning process. The primary motivation is to make programming an

activity that is accessible to the hands and minds of younger children by making it more direct and less abstract. Tangible programming may have an appeal even to experienced abstract thinkers.

### **Collaborative Programming**

One of my original motivations for pursuing tangible programming is to design a programming environment where small group children can build programs together. Like Suzuki and Kato, I am very much interested in programming environments which encourages teamwork. This is difficult using traditional screen-based programming environments because only one user can type on the keyboard at a time. When a program is constructed out of physical objects i.e., in the tangible way, several children sitting around a table can work together to integrate or modify the program as a team or each person can create or modify their own methods independently of their teammates.

### **Debugging**

The problematic part of learning how to program is learning how to debug programs. Debugging is a difficult activity to teach beginners, because the process of debugging is largely a process driven by knowing what to look for and having an extremely clear idea of what is going on inside the computer. Since it is exactly this modeling of these invisible blocks that one is trying to teach when teaching students how to program, learning how to program is hard. One of my goals is to make these invisible blocks visible by making programs something that you can watch as it runs, thereby making debugging a skill with a lower learning curve.

### **Limitations of “visual” programming languages**

Screen-based graphical programming languages suffer from a numerous limitation; Tools for manipulating textual-based programming languages are much more mature than graphical programming tools. Textual programming languages make better use of screen

real estate than graphical programming languages, which often include extra decorations around each functional block. In the domain of programming languages for children, this is less of an issue because novice programmers need to gain experience while writing simple programs before they start writing larger, more complex programs. A common approach for evaluating and comparing visual languages is to quantify how sophisticatedly and briefly an algorithm can be implemented.

### CHAPTER III: RELATED WORK

Research studies about tangible programming languages using tangible programming interface influenced us to implement and teach programming syntax to the young school students. The earliest and most popular tangible programming language is Suzuki and Kato's AlgoBlocks [1], which looks identical to the Logo programming with interlocking aluminum blocks containing programming syntax on it. These aluminum blocks are integrated to form programs.

Most recently, Horn and Jacob [2] developed tangible programming language called Tern, which emphasized on designing an inexpensive, durable, and practical system for classroom use. It consists of a collection of wooden blocks which looks alike to the pieces of Jigsaw puzzle. Students integrate these pieces to form computer programs which may include action, commands, loops, branches, and subroutines.

In a similar project, Horn, Solovey, and Jacob [3] at Tufts University designed an initial evaluation of a tangible computer programming for young children on display at the Boston Museum of Science. The results from their evaluation at the science museum indicate that passive tangibles can preserve many of the pros of tangible interaction for informal science learning while remaining cost-effective and reliable.

Zukerman and Resnick's [4] system blocks project that provides a physical tangible interface for kids to model and explore dynamic systems. These system blocks with embedded electronics express simple behaviors in a system. Their hope is that System Blocks will enable children younger than sixth grade to model, simulate and analyze dynamic systems that are meaningful to them.

McNerney's [5] article from Springer research at MIT developed a tangible programming bricks system- a platform for creating microworlds that help children to explore computation and lateral thinking through free-form play.

Wyeth and Purchase [6] from the University of Queensland have designed Electronic Blocks are tangible programming elements – blocks can be stacked and arranged to form structures that interact with the physical world. By stacking Electronic Blocks, young children build “computer programs” where each stack of Electronic Blocks is capable of a different function.

Wellner [7] at Cambridge Laboratory has written a technical report which describes the thresholding problem which must be overcome by Digital Desk applications and have developed a quick adaptive thresholding algorithm that has proven to be quite suitable for current purposes and which probably can be implemented in hardware.

Blackwell and Hague [8] at University of Cambridge created a novel programming language, Media Cubes, which supports end-user programming for domestic contexts. Media Cubes are wooden blocks with bidirectional, infrared communication capabilities. The Induction coil antenna detects direct proximity of another cube and is also used to establish a relationship between an appliance and a cube.

de Ipina, Mendonca and Hopper [9] at University of Cambridge have developed TRIP (Target Recognition using Image Processing) which is a novel cost-effective and easily deployable sensor technology that offers an excellent degree of accuracy and performance for the identification and 3-D location of tagged entities.

Finally, Maloney, Burd, Kafai, Rusk, Silverman, and Resnick [10] from Lifelong Kindergarten Group have developed an educational language called Scratch, which is not tangible and adds programmability to the media-rich and networked-based activities that are most popular among youth to further the development of technological applications at after-school centers in economically-disadvantaged communities.

In most of the above examples, the wooden blocks or bricks that make up the programming language contains some form of nested electronic components. When these

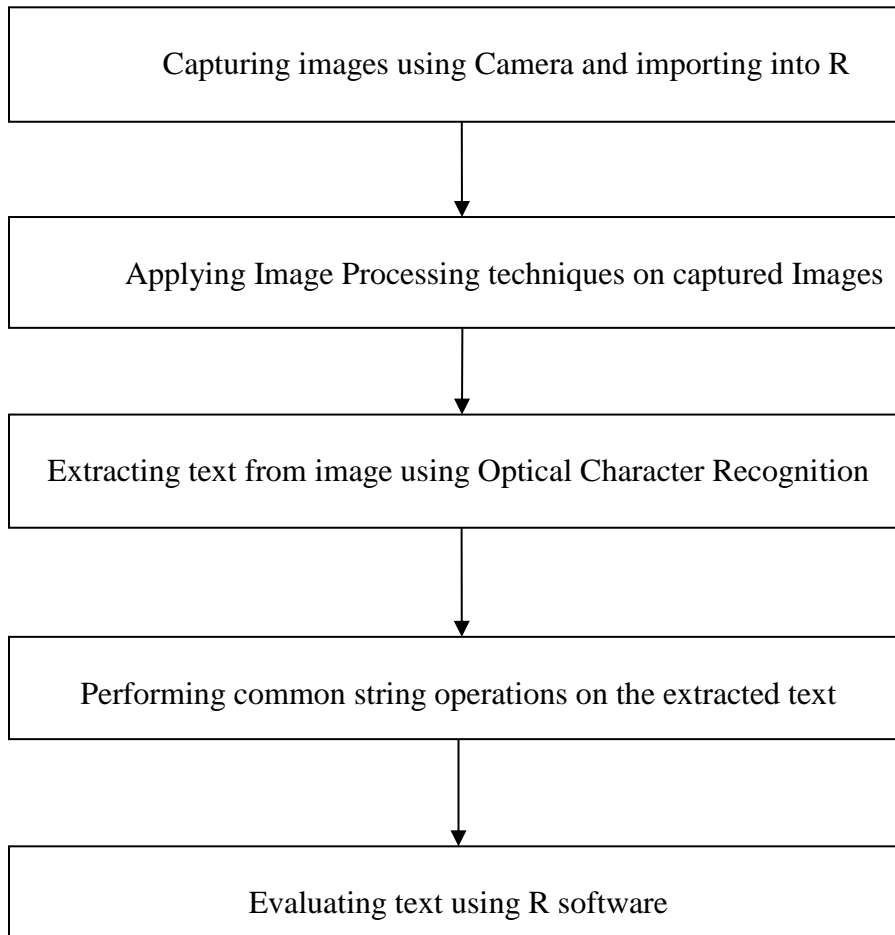
blocks are integrated, they provide dynamic behavior by sequentially executing algorithms through integrating blocks. Our model differs from the experiments on a tangible programming. It is a ubiquitous technique for implementing and teaching tangible programming using Image processing techniques in R programming for middle and late elementary school students. It emphasizes the use of inexpensive and durable blocks with no embedded electronic supplies. Students integrate these wooden blocks to create a computer program in offline settings such as on desks and use R software to compile their code. The programming languages explained through this approach at an early age offers more retention rate of syntax and semantics of the programming than traditional screen-based classroom teaching.

## CHAPTER IV: METHODOLOGY

Here, the overall workflow of the thesis is explained in the Figure 4.1. Initially the students integrate the colored wooden blocks which contains R syntax and the image are captured using phone and are imported into R software, then various image processing techniques are applied on the captured images and then the text is extracted using Optical Character Recognition (OCR) and various string operations are performed on the evaluated text is evaluated by eval function.

*Figure 4.1*

### *Overall workflow*

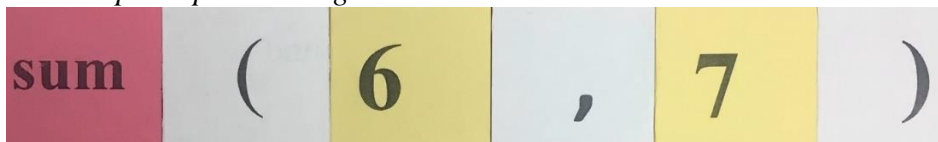


### Capturing the images using Camera

Students integrate the wooden cubes on the floor or a desk and then the image representation are captured using a Camera. A java application controls the flash, optical zoom, and image resolution. Captured images are saved as JPEG type on the file system and are imported into R programming software.

*Figure 4.2*

*An example captured image*



### Wooden blocks

The prototype uses tangible wooden blocks. These blocks are 2 X 2 X 2 dimensions in length, breadth and height respectively. The R programming syntax are imprinted on the colored papers and are pasted on the wooden blocks. The style of the text is Times New Roman for the text on all the cubes and font of the text varies depending on the programming syntax.

*Table 4.1*

*Syntax and cube color*

Syntax	Cube Color
Function	pink
variables	blue
operators	green
numbers	yellow
Special characters	white

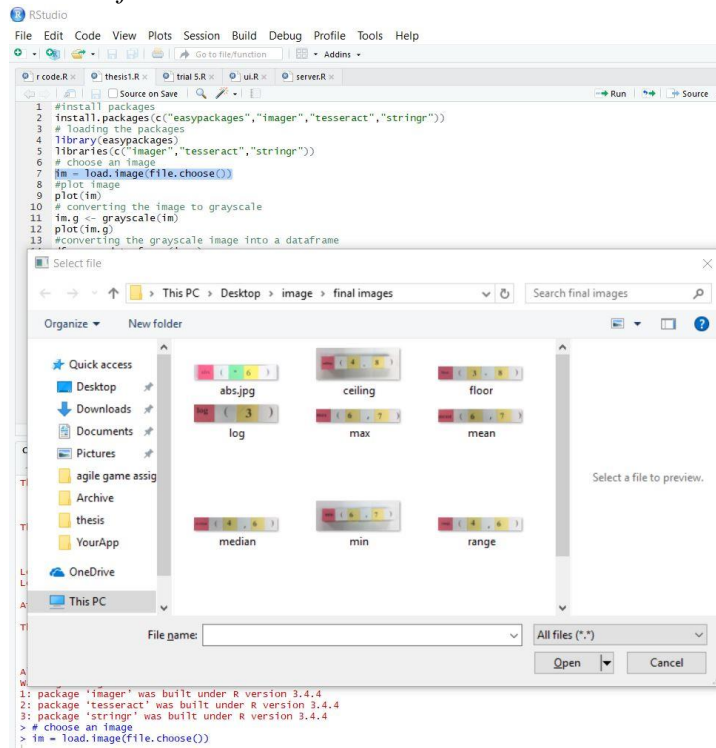
The numbers, variables, special characters and operators have the font size of 85 points whereas the functions have 54 points except the round function which is 36 points. Based on the syntax the colored papers are chosen. The colored paper for the programming syntax is very essential because it makes the children to integrate the blocks very easily. There is a rule while integrating the wooden cubes i.e., no two cubes which has the same color should not be integrated together.

### Importing the images into R software

The images are imported into R studio via the command line. Whenever the command is executed on the R interpreter a pop-up box shows up asking you to select the image and when the image is selected by the user and clicked open, the image gets imported into R software. The Figure 4.3 shows importing an image into R software.

*Figure 4.3*

#### *Importing an image into R software*



## **Applying image processing techniques on the captured images using imager package**

The various image processing techniques are applied to captured image by using a “imager” package.

### **Overview of imager package**

The imager package in R makes the image processing work much easier. It is based on CImg, a C++ library which provides an easy-to-use and consistent Application Programming Interface(API) for image processing, which imager largely replicates. CImg supports images in up to four dimensions, which makes it suitable for basic video processing/hyperspectral imaging as well. The imager aims to be fast, but also R-friendly, and defines many convenience functions that make it easy to work with native R datatypes and functions.

### **Grayscale image**

A greyscale image is only one image which contains shades of grey color. The reason for differentiating grey scale image from color image is that fewer information needs to be provided for each pixel. In fact, a ‘grey’ color is one in which the red, green and blue components have identical intensity in RGB space, and so it is essential to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full color image.

Frequently, the greyscale intensity is stored as 8-bit integer which gives 256 possible different shades of gray from black to white. If the levels are evenly spaced then the differences between successive graylevels is meaningfully better than the grayscale resolving power of the human eye.

Greyscale images are most common because most of the today's display and image capture hardware can only support 8-bit images. In addition, grayscale images are

completely sufficient for most of the tasks and so there is no need to use more complicated and harder-to-process color images.

## Thresholding

Thresholding corresponding to setting all values below a threshold to 0, all above to 1. If you call `threshold` with `thr="auto"` a threshold will be computed automatically using `kmeans`. This works well if the pixel values have a clear bimodal distribution. If you call `threshold` with a string argument of the form `"XX%"` (e.g., `"98%"`), the threshold will be set at percentile `XX`. Computing quantiles or running `kmeans` is expensive for large images, so if `approx == TRUE` `threshold` will skip pixels if the total number of pixels is above 10,000. Note that thresholding a color image will threshold all the color channels jointly, which may not be the desired behavior.

## Pixel

The Pixel is the smallest element of an image. Each pixel corresponds to any one value. In a gray scale image, the value of the pixel is between 0 and 255. The value of the pixel at any point correspond to the intensity of the light photons striking at that point.

*Table 4.2*

*Threshold value for respective function image*

Function image	Threshold value
Minimum	5
Square root	6
Sum	4.75
Maximum	5.5
Log	5.5
Round	11.40
Mean	2.5

Since each imported image has different number of pixels and different illumination of light, the threshold value of every image varies. The above Table 2 shows the various threshold values and respective function.

### **Shrink and grow pixels**

The Grow/shrink a pixel set through morphological dilation/erosion. The default is to use square or rectangular structuring elements, but an arbitrary structuring element can be given as input. A structuring element is a pattern to be moved over the image:

For Example: A 3\*3 square.

In “shrink” mode, an element of the pixset is retained if and only if the structuring element fits entirely within the pixset. In “grow” mode, the structuring element acts like a neighbourhood: all pixels that are in the original pixset “or” in the neighbourhood defined by the structuring element belong to the new pixset.

*Table 4.3*

*Shrink and grow values for respective function image*

<b>Function image</b>	<b>Shrink and grow value</b>
Minimum	3
Square root	5
Sum	10
Maximum	11
Log	12
Round	2
Mean	7

### **Extracting text from images using Optical Character Recognition**

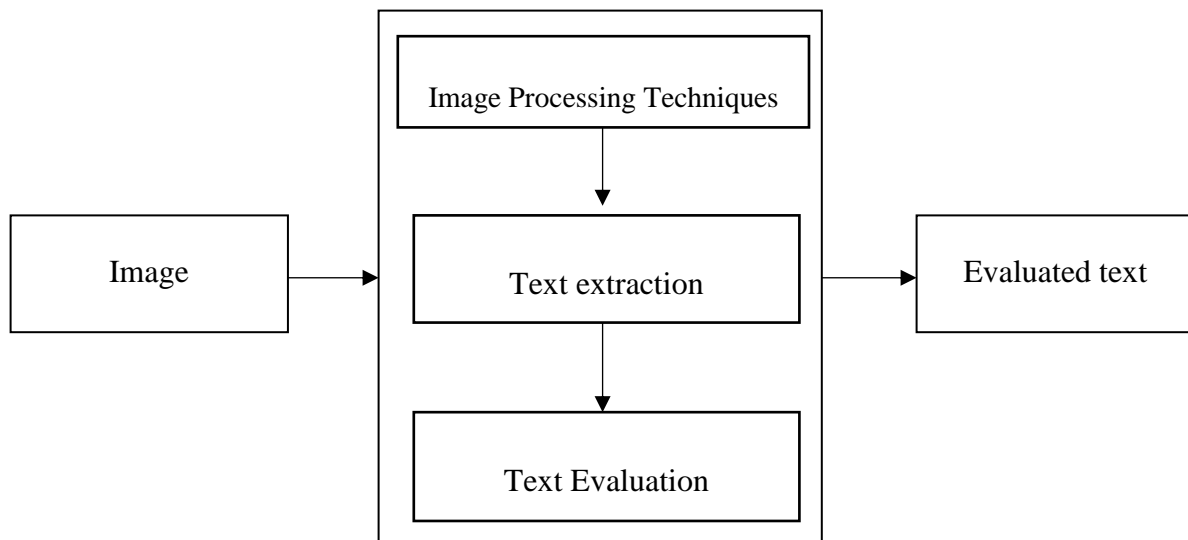
Text extraction from the processed image is performed using a package called “Tesseract” which is an open source (Optical Character Recognition) OCR Engine for R. Optical character recognition (OCR) is the process of extracting written or printed text from the images such as photos and scanned documents into machine-encoded text. It uses training data for the language you are reading to perform OCR and works best for images with high contrast, little noise and horizontal text.

### **Evaluating the extracted text using R software**

Finally, the evaluation of the extracted text is performed by using a function called “eval” which is available in R programming language.

*Figure 4.4*

*Input and Output to R software*



In the above Figure 4.4, the input is a JPEG or PNG image which is imported into R Software and image processing techniques, text extraction and text evaluation is performed. The output of the R software is the evaluated text.

## **Automating the tangible programming using the User-interface**

### **User interface**

The user interface is designed using html, CSS, JavaScript, and R software. It contains the drag and drop elements, buttons. The student or the user drags and drops the elements and clicks on the submit button. The syntax written by the user is evaluated using the R compiler and displayed on the user interface. To make the user interface easy accessible for the students, it is hosted on the shiny server and is accessible using the single link.

## CHAPTER V: EXPERIMENTAL AND RESULTS

Here, the results that is the final output where the text is detected and extracted from the image and evaluated in R Programming language software for the output by using the methodology as explained in earlier chapters.

### Minimum function

The Figure 5. 1 shows A minimum function colored images which is captured using phone and imported into R software and displayed.

*Figure 5.1*

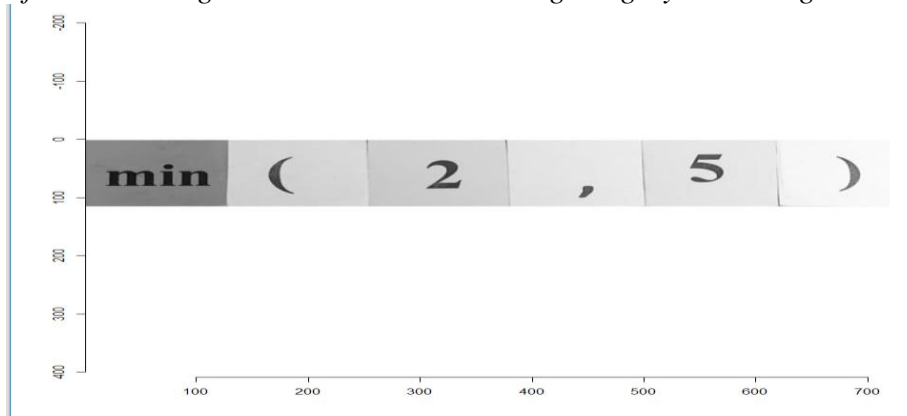
*A colored image which contains minimum function*



The Figure 5. 2 shows the application of an image processing technique on the minimum function colored image to greyscale image using imager package in R software.

*Figure 5.2*

*After converting the minimum colored image to greyscale image*



The Figure 5. 3 shows the conversion of minimum function greyscale image to data frame in R

*Figure 5.3*

*After converting the greyscale image into data frame*

```
> im = load.image(file.choose())
> display(im, rescale = TRUE)
> plot(im)
> im.g <- grayscale(im)
> plot(im.g)
> df <- as.data.frame(im.g)
> head(df, 5)
  x y value
1 1 1 0.5694902
2 2 1 0.5734118
3 3 1 0.5773333
4 4 1 0.5734118
5 5 1 0.5655686
>
>
```

The Figure 5. 4 shows the application of a linear model on the minimum function greyscale image data frame

*Figure 5.4*

*After applying the linear model on the minimum function greyscale image data frame*

```
> m <- lm(value ~ x + y, data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.70980 -0.03107  0.01236  0.06036  0.23635

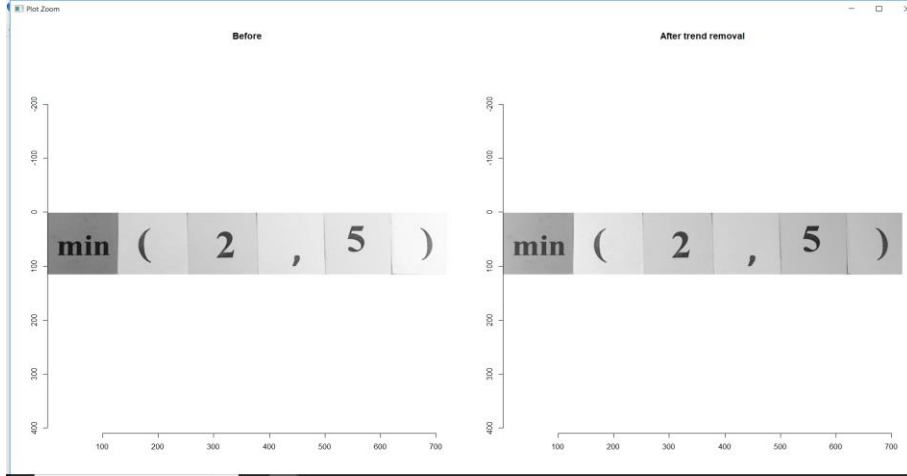
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.090e-01  1.149e-03  529.99  <2e-16 ***
x             5.793e-04  2.083e-06  278.04  <2e-16 ***
y            -2.054e-04  1.303e-05  -15.77  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1243 on 82682 degrees of freedom
Multiple R-squared:  0.484,    Adjusted R-squared:  0.484
F-statistic: 3.878e+04 on 2 and 82682 DF,  p-value: < 2.2e-16
```

The Figure 5. 5 shows the plot before and after trend removal from the minimum function greyscale image

*Figure 5.5*

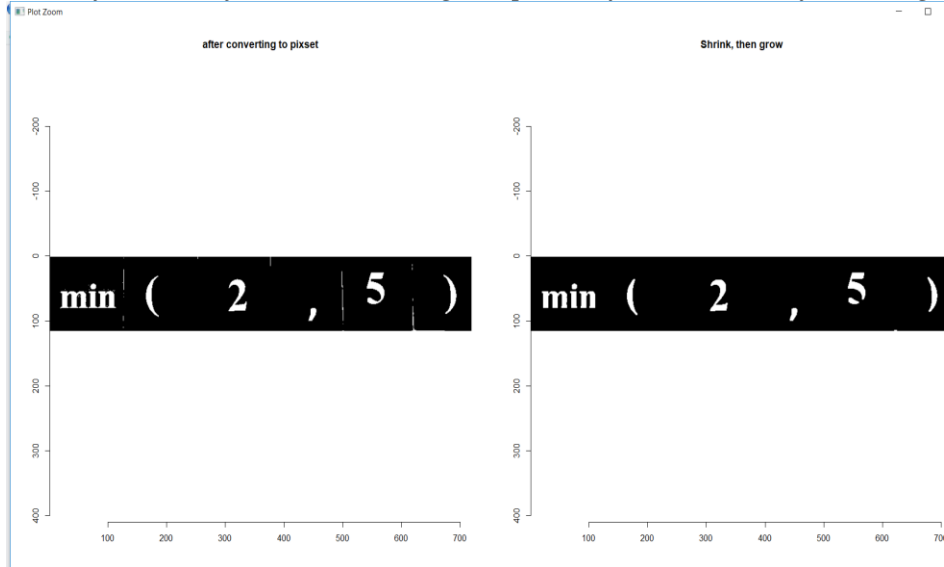
*Plot before and after trend removal from the greyscale minimum function image*



The Figure 5. 6 shows the plot before and after shrink, then grow pixels of the minimum function greyscale image.

*Figure 5.6*

*Plot before and after shrink, then grow pixels of the minimum function greyscale image*



The Figure 5. 7 shows the number of pixels in a minimum function greyscale image

*Figure 5.7*

*Displaying the number of pixels in a minimum function greyscale image*

```
>
> im.t <- threshold(im.f,"%5")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,3) %>% grow(3)
> plot(px1, main = "shrink, then grow")
> px1
Pixel set of size 3973. width: 719 pix Height: 115 pix Depth: 1 Colour channels: 1
> |
```

The Figure 5. 8 shows the extracting text from the minimum function greyscale image using optical character recognition and applying the string operations on the extracted text using stringr package. The output after applying the string operation is evaluated using eval function and display the results.

*Figure 5.8*

*Extracting the text from image and evaluation*

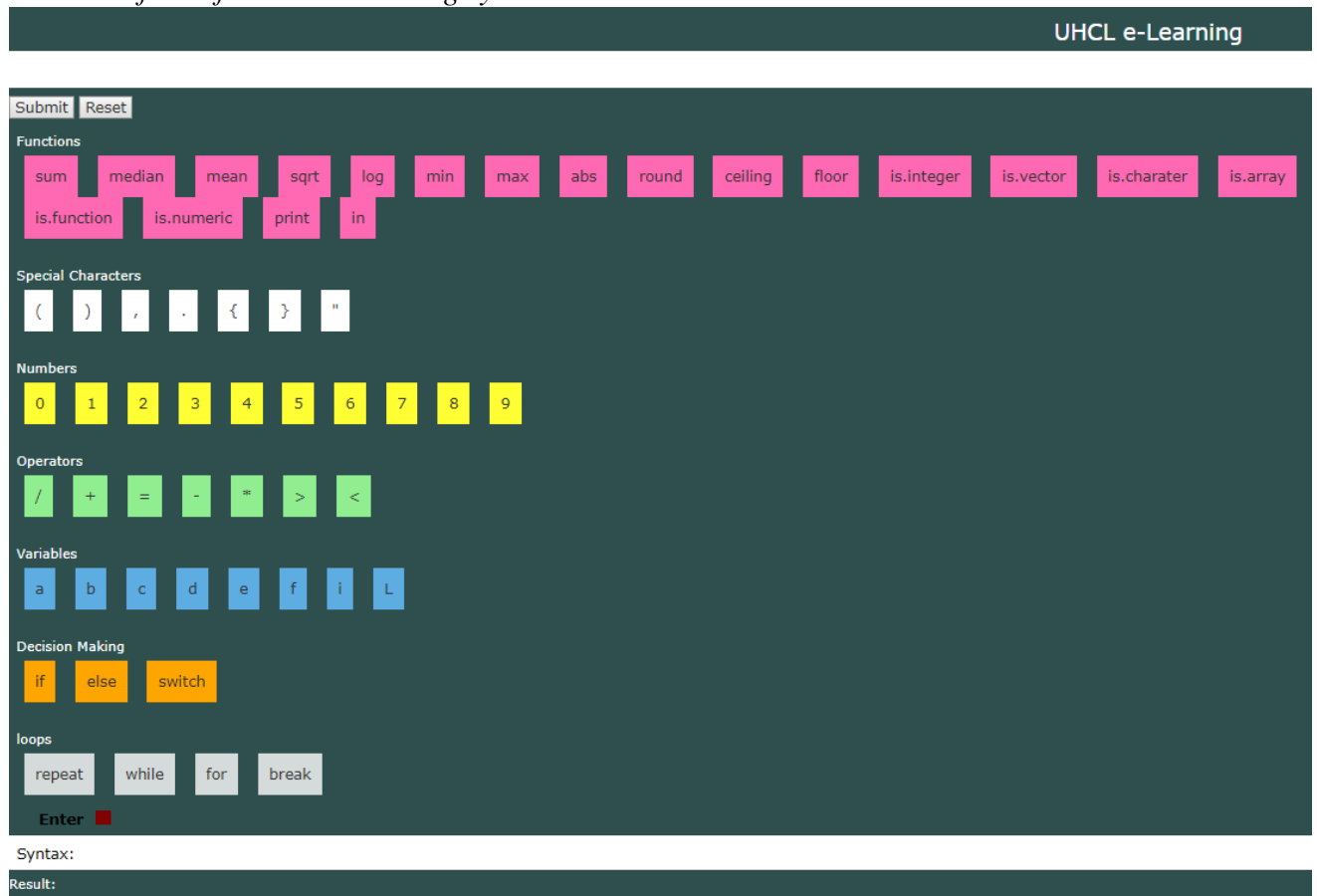
```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
warning: Invalid resolution 0 dpi. Using 70 instead.
> data1 <- str_replace_all(data," ","")
> cat(data1)
min(2,5)

> eval(parse(text=data1))
[1] 2
> |
```

Alternatively, the programming can also be learnt through the Interface. The below Figure 5.9 shows the user interface

*Figure 5.9*

*User Interface of UHCL e-learning system*



The Figure 5.10 shows the calculating the sum of 6 numbers. Whenever user drags and drops the elements and click the submit button the result is displayed.

*Figure 5.10*

*Calculating the sum of 6 numbers*

UHCL e-Learning

sum ( 2 , 5 , 6 , 3 , 9 , 4 )

Submit Reset

Functions

sum median mean sqrt log min max abs round ceiling floor is.integer is.vector is.charater is.array is.function is.numeric print in

Special Characters

( ) , . { } "

Numbers

0 1 2 3 4 5 6 7 8 9

Operators

/ + = - \* > <

Variables

a b c d e f i L

Decision Making

if else switch

Enter

Syntax:sum(2,5,6,3,9,4)

Result:

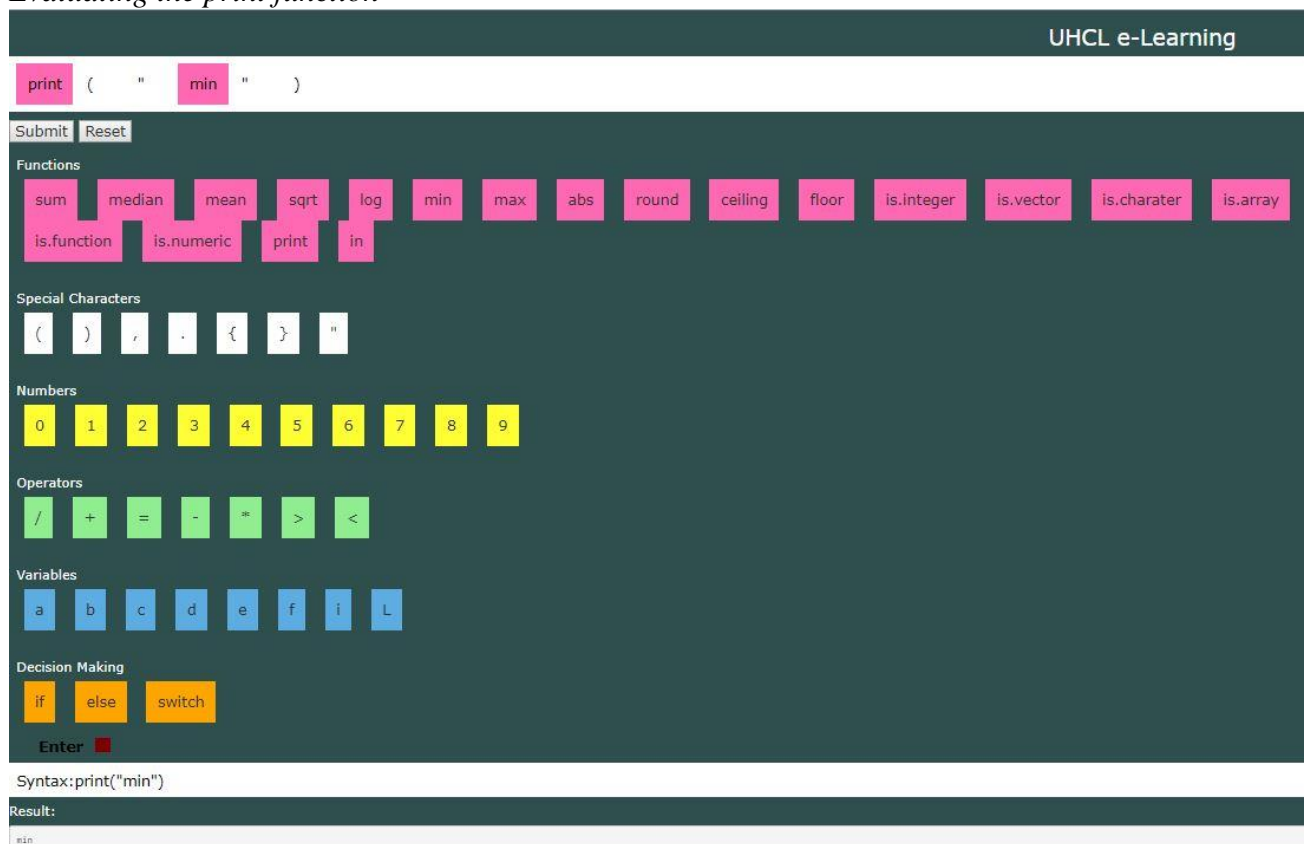
29

The Figure 5.11 shows the print function. The text passed to the print function will be displayed as a result to the user.

For Example: In the below Figure 5.11, the text “min” is passed to the print function. When the user clicks the submit button, the text min is displayed as an output and when the user clicks the Reset button, the interface will be cleared.

*Figure 5.11*

#### *Evaluating the print function*



The Figure 5.12 shows the passing the value to the variable and displaying it.

For Example: In the Figure 5.12 the value “2” is passed to the variable “a” and displayed to the user.

*Figure 5.12*

*Passing the value to the variable and displaying*

UHCL e-Learning

a = 2 a

Submit Reset

**Functions**  
sum median mean sqrt log min max abs round ceiling floor is.integer is.vector is.charater is.array  
is.function is.numeric print in

**Special Characters**  
( ) / . { } "

**Numbers**  
0 1 2 3 4 5 6 7 8 9

**Operators**  
/ + = - \* > <

**Variables**  
a b c d e f i L

**Decision Making**  
if else switch

Enter

Syntax: a=2 a

Result:  
2

The Figure 5.13 shows evaluating the decision making “if statement” and displaying it on the user interface.

For Example: The integer value “3L” is passed to the variable “a”. The decision making if statement is evaluated for is.integer. If the statement is evaluated true then it prints “a”.

*Figure 5.13*

*Evaluating the if statement and displaying*

The screenshot displays the UHCL e-Learning interface. At the top right, the text "UHCL e-Learning" is visible. Below this is a code editor with a dark background. The code entered is: `a = 3L if ( is.integer ( a ) ) { print ( " a " ) }`. The code is color-coded: variables are blue, operators are green, and functions/keywords are pink. Below the code editor are buttons for "Submit" and "Reset".

Below the code editor is a panel with various categories of elements:

- Functions:** sum, median, mean, sqrt, log, min, max, abs, round, ceiling, floor, is.integer, is.vector, is.charater, is.array, is.function, is.numeric, print, in.
- Special Characters:** (, ), /, ., {, }, "
- Numbers:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Operators:** /, +, =, -, >, <
- Variables:** a, b, c, d, e, f, i, L
- Decision Making:** if, else, switch

At the bottom of the panel is an "Enter" button with a red square icon.

Below the panel, the syntax is shown: `Syntax: a=3L if(is.integer(a)){print("a")}`.

At the bottom, the "Result:" section shows the output: `a`.

The Figure 5.14 shows evaluating the decision making “switch statement” and displaying it on the user interface.

For Example: The variable “a” is passed as a first element to the switch decision making statement and printing the variable a on the user Interface.

*Figure 5.14*

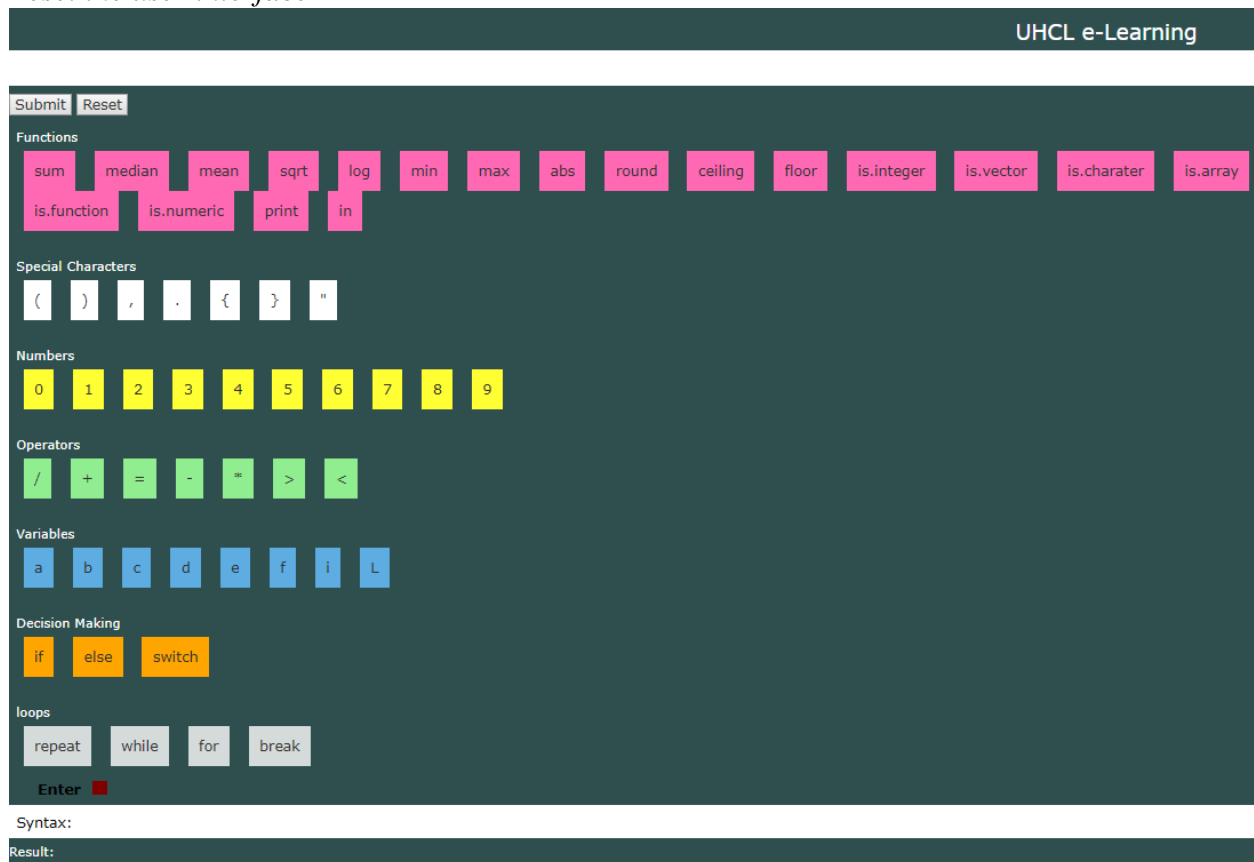
*Evaluating the switch statement and displaying*

The screenshot displays the UHCL e-Learning interface. At the top right, the text "UHCL e-Learning" is visible. Below this, a code editor shows the following code: `a = switch ( 1 , " a " ) { print ( a ) }`. The code is color-coded: variables are blue, operators are green, and strings are yellow. Below the code editor, there are buttons for "Submit" and "Reset". The interface is divided into several sections: "Functions" (sum, median, mean, sqrt, log, min, max, abs, round, ceiling, floor, is.integer, is.vector, is.charater, is.array, is.function, is.numeric, print, in), "Special Characters" ( ( , ) , . , { , } , " ), "Numbers" (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), "Operators" ( / , + , = , - , \* , > , < ), "Variables" (a, b, c, d, e, f, i, L), and "Decision Making" (if, else, switch). At the bottom, there is an "Enter" button. Below the code editor, the syntax is shown: `Syntax: a=switch(1,"a") print(a)`. The "Result:" section shows the output: `a`.

The Figure 5.15 shows resetting the user interface by clicking on the reset button.

Figure 5.15

*Reset the user interface*



## VI. CONCLUSION AND FUTUREWORK

### **Conclusion**

This thesis intends to explain a ubiquitous technique for designing and learning tangible programming using manually through image processing techniques and automatically using the user interface. These introduce the programming language at an early age for late elementary and middle school students. Our tangible programming language consists of inexpensive and durable wooden cubes, which have no embedded power supplies. The programs would be written by students on a physical desk or a floor and then the pieces would be integrated to form a computer program [4]. The programs would then compile in R software using image processing techniques. This practical approach expects to help students to learn programming in their classrooms with minimal tools (limited desktops or laptop computers) and limited adult guidance.

Alternatively, the students or the user can also learn programming automatically through the user interface designed in R using shiny package. Whenever user drags and drops the elements and click the submit button, the results are evaluated in R software and displayed on the user interface.

### **Future Work**

Firstly, this research can be used to create an accurate machine learning models capable of localizing and identifying multiple objects in a single image. After detecting the objects, the text can extracted by identifying the labels and can evaluated using R software for the result.

Secondly, stencils can be used for writing programming syntax and to recognize, extract and evaluate using the interpreter or the compiler for results

Finally, magnetic blocks could be used instead of wooden blocks to recognize the programming syntax. The magnetic blocks are designed such that, two same-colored

magnetic blocks would not be attracted. When the last magnetic block is integrated, the mobile app could automatically capture the image and give the result.

## REFERENCES

1. Suzuki, H. and Kato, H. Interaction-level support for collaborative learning: Algoblock—an open programming language. In Proc. CSCL '95, Lawrence Erlbaum (1995).
2. Horn, M. and Jacob, R.J.K. Tangible Programming in the Classroom: A Practical Approach. Extended Abstracts CHI 2006, ACM Press (2006).
3. Horn, M. S., Solovey, E. T., & Jacob, R. J. K. (2008). *Tangible programming and informal science learning: making TUIs work for museums*. Paper presented at the Proceedings of the 7th international conference on Interaction design and children, Chicago, Illinois.
4. Zuckerman, O. and Resnick, M. A physical interface for system dynamics simulation. Extended Abstracts CHI 2003, ACM Press (2003), 810-811.
5. McNerney, T.S. From turtles to Tangible Programming Bricks: explorations in physical language design. Personal Ubiquitous Computing, 8(5), Springer Verlag (2004), 326–337.
6. Wyeth, P. and Purchase, H.C. Tangible programming elements for young children. Extended Abstracts CHI 2002, ACM Press (2002), 774–775.
7. Wellner, P.D. Adaptive thresholding for the DigitalDesk. Technical Report EPC-93-110, Euro PARC (1993).
8. Blackwell, A.F. and Hague, R. Autohan: An architecture for programming in the home. In Proc. IEEE Symposia on Human-Centric Computing Languages and Environments 2001, 150-157.
9. de Ipina, D.L., Mendonca, P.R.S. and Hopper, A. TRIP: A low-cost vision-based location system for ubiquitous computing. Personal and Ubiquitous Computing, 6 (2002), 206–219.

10. Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M. Scratch: a sneak preview. In Proc. Second International Conference on Creating, Connecting, and Collaborating through Computing C5 '04. IEEE (2004), pp 104-109.
11. Mayer, R. (1981). The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys (CSUR)*, 13(1), 121-141. doi:10.1145/356835.356841.
12. Pane, J. F., Ratanamahatana, C., & Myers, B. A. (2001). Studying the language and structure in non-programmers' solutions to programming problems. *INTERNATIONAL JOURNAL OF HUMAN-COMPUTER STUDIES*, 54(2), 237-264.
13. Pea, R. D., & Kurland, D. M. (1984). ON THE COGNITIVE EFFECTS OF LEARNING COMPUTER-PROGRAMMING. *NEW IDEAS IN PSYCHOLOGY*, 2(2), 137-168.
14. Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), 137-172. doi:10.1076/csed.13.2.137.14200.
15. R.J.K. Jacob. "CHI 2006 Workshop Proceedings: What is the Next Generation of Human-Computer Interaction?," Technical Report 2006-3, Dept. of Computer Science, Tufts University, Medford, Mass. (2006).

## APPENDIX A: OTHER EXPERIMENTAL RESULTS

### Square root Function

Figure A.1

*A colored image which contains square root function*



Figure A.2

*After converting the square root image into greyscale image*

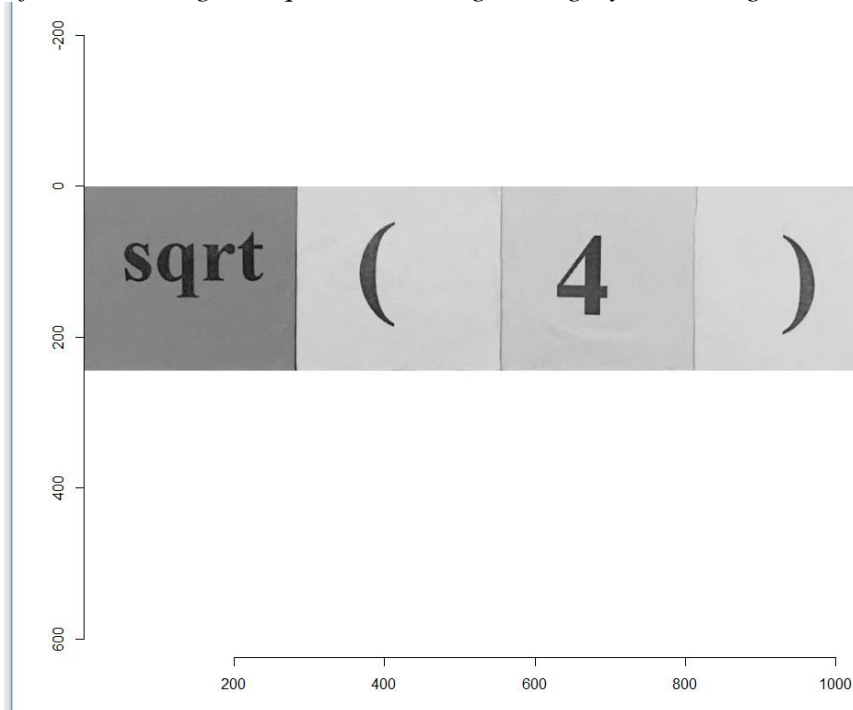


Figure A.3

*After converting the greyscale image into data frame*

```
> im = load.image(file.choose())
> display(im, rescale = TRUE)
> plot(im)
> im.g <- grayscale(im)
> plot(im.g)
> df <- as.data.frame(im.g)
> head(df, 5)
  x y value
1 1 1 0.6364706
2 2 1 0.6364706
3 3 1 0.6325490
4 4 1 0.6286275
5 5 1 0.6247059
```

Figure A.4

*Applying linear model on the data frame of square root function greyscale image*

```
> m <- lm(value ~ x + y, data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.58398 -0.03394 -0.00425  0.07582  0.37142

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.207e-01  5.585e-04 1111.32  <2e-16 ***
x             2.937e-04  7.126e-07  412.11  <2e-16 ***
y            -5.126e-05  2.978e-06  -17.21  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1055 on 250877 degrees of freedom
Multiple R-squared:  0.4041,    Adjusted R-squared:  0.4041
F-statistic: 8.506e+04 on 2 and 250877 DF,  p-value: < 2.2e-16
```

Figure A.5

*Before and after trend removal from square root function greyscale image*

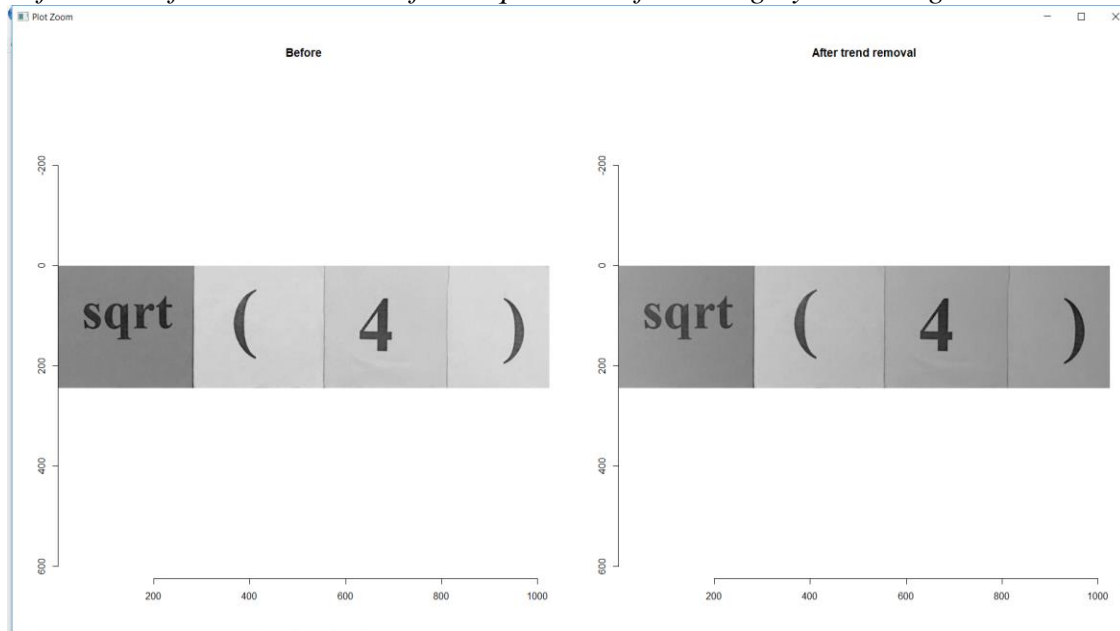


Figure A.6

*Before and after shrink, then grow pixels of square root function greyscale image*

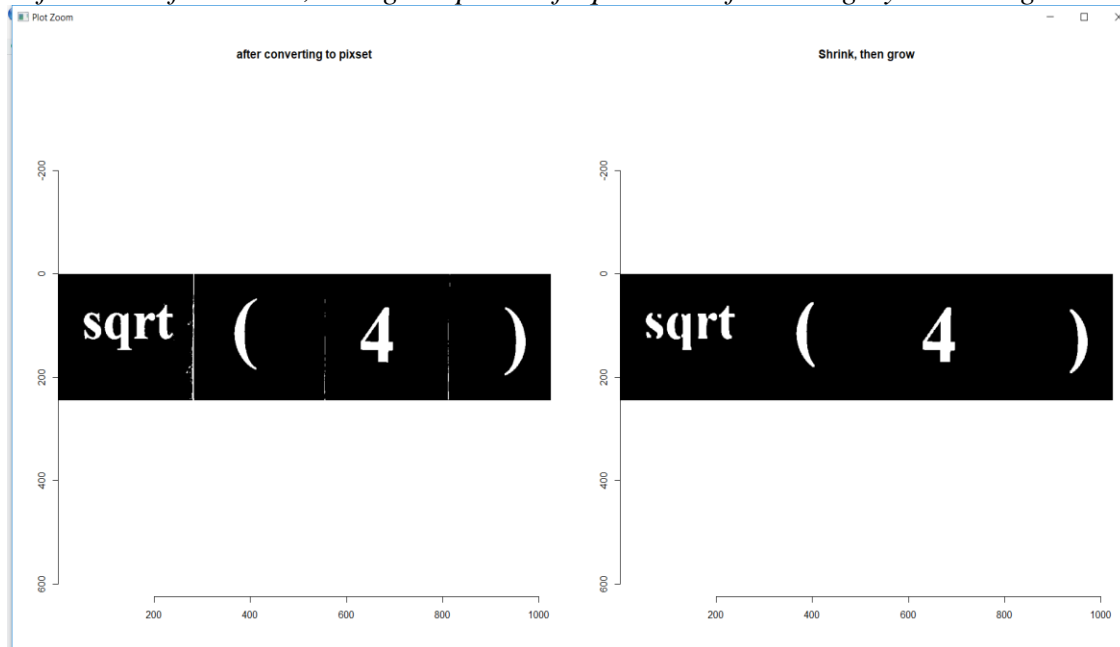


Figure A.7

*Displaying the number of pixels in a square root function grey scale image*

```
> layout(t(1:2))
> im.f <- im.g-fitted(m)
> plot(im.g,main="Before")
> plot(im.f,main="After trend removal")
> im.t <- threshold(im.f,"%6")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,5) %>% grow(5)
> plot(px1, main = "shrink, then grow")
> px1
Pixel set of size 12224. Width: 1024 pix Height: 245 pix Depth: 1 Colour channels: 1
> |
```

Figure A.8

*Extracting and evaluating text from square root function greyscale image*

```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
warning. Invalid resolution 0 dpi. Using 70 instead.
> data1 <- str_replace_all(data," ","")
> cat(data1)
sqrt(4)

> eval(parse(text=data1))
[1] 2
```

## Sum Function

Figure A.9

*A colored image which contains sum function*

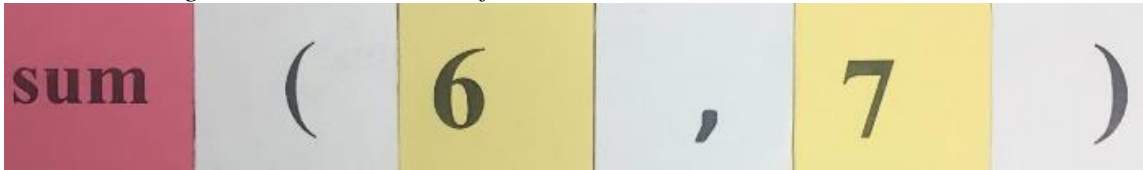


Figure A.10

*After converting the sum function colored image into greyscale image*

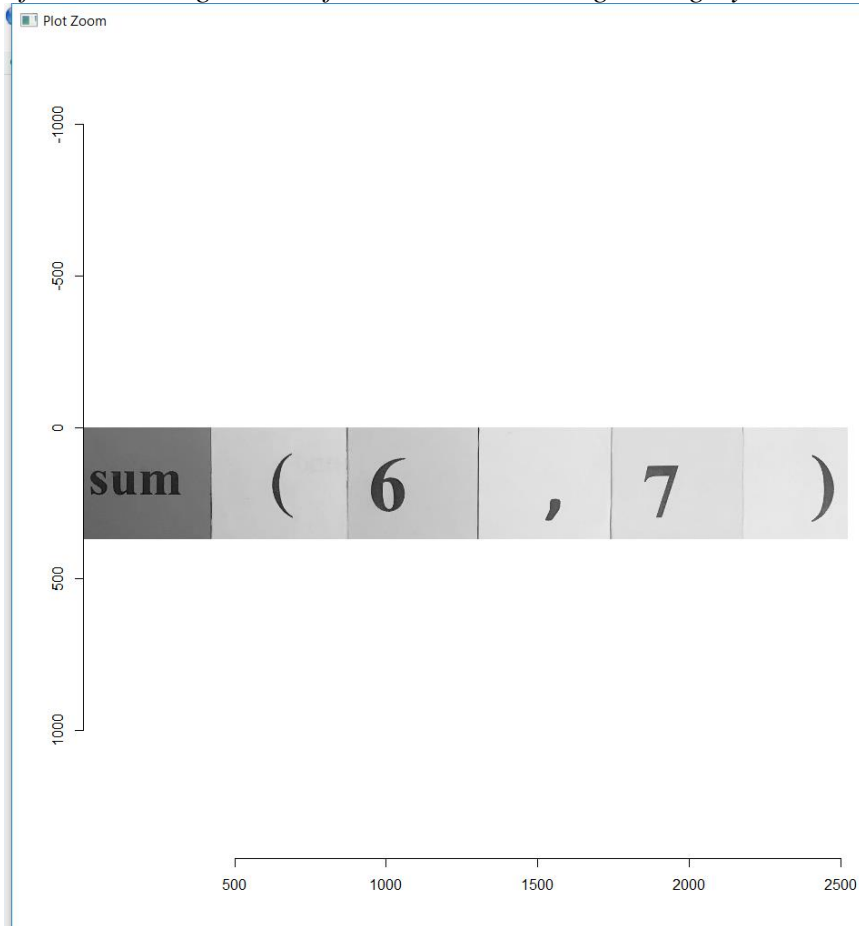


Figure A.11

After converting the sum function greyscale image into data frame

```
Console Terminal x
~/
> df <- as.data.frame(im.g)
> head(df,5)
  x y value
1 1 1 0.4795294
2 2 1 0.4834510
3 3 1 0.4834510
4 4 1 0.4795294
5 5 1 0.4834510
.
```

Figure A.12

Applying linear model on the data frame of sum function greyscale image

```
> m <- lm(value ~ x + y, data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.70131 -0.04726  0.03443  0.07774  0.24523

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.782e-01  3.407e-04 1697.01  <2e-16 ***
x             1.630e-04  1.766e-07  922.91  <2e-16 ***
y            -5.124e-05  1.207e-06  -42.45  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1241 on 930984 degrees of freedom
Multiple R-squared:  0.4783,    Adjusted R-squared:  0.4783
F-statistic: 4.268e+05 on 2 and 930984 DF,  p-value: < 2.2e-16
```

Figure A.13

*Before and after trend removal from sum function greyscale image*

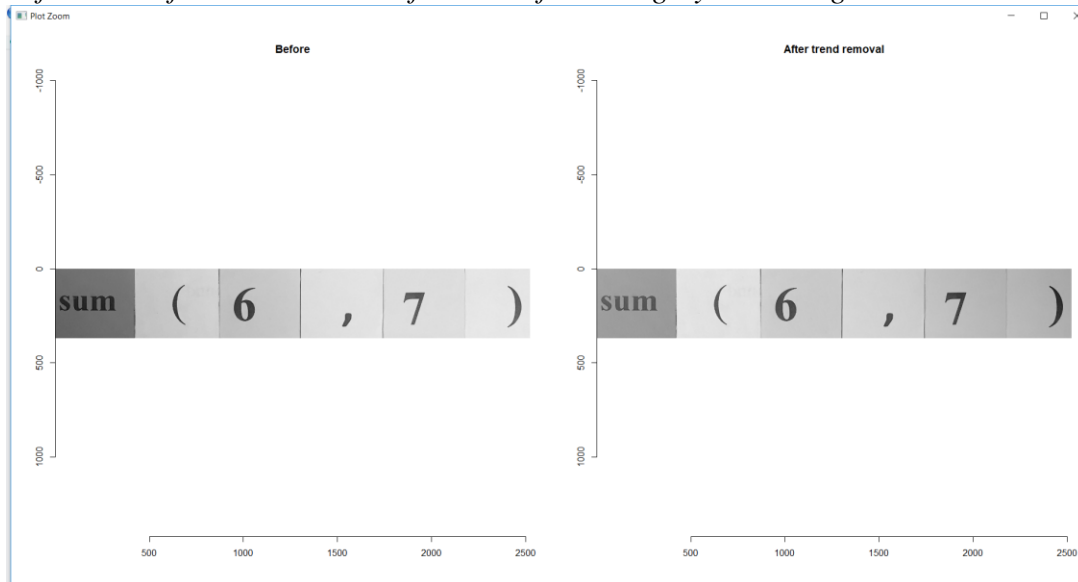


Figure A.14

*Before and after shrink, then grow pixels from sum function greyscale image*

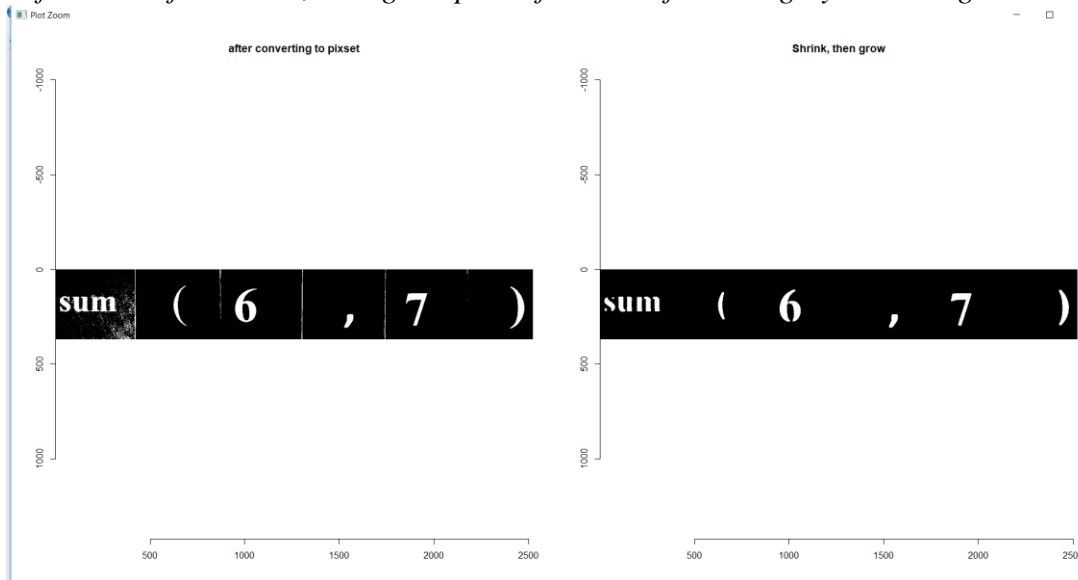


Figure A.15

*Displaying the number of pixels in a sum function grey scale image*

```
> layout(t(1:2))
> im.f <- im.g-fitted(m)
> plot(im.g,main="Before")
> plot(im.f,main="After trend removal")
> im.t <- threshold(im.f,"%4.75")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,10) %>% grow(10)
> plot(px1, main = "Shrink, then grow")
> px1
Pixel set of size 37849. width: 2523 pix Height: 369 pix Depth: 1 Colour channels: 1
```

Figure A.16

*Extracting and evaluating text from sum function greyscale image*

```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
warning. Invalid resolution 0 dpi. using 70 instead.
> data1 <- str_replace_all(data," ","")
> cat(data1)
sum(6,7)

> eval(parse(text=data1))
[1] 13
```

## Maximum Function

Figure A.17

*A colored image which contains maximum function*

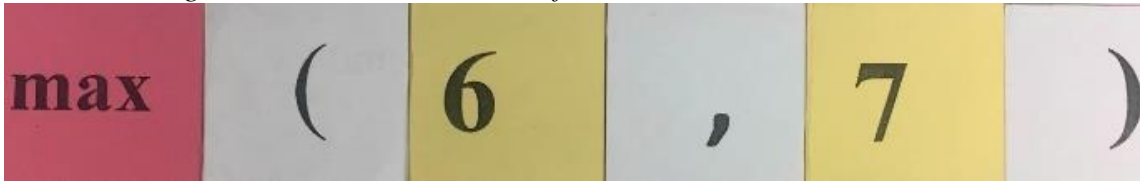


Figure A.18

*After converting the maximum function image into greyscale image*

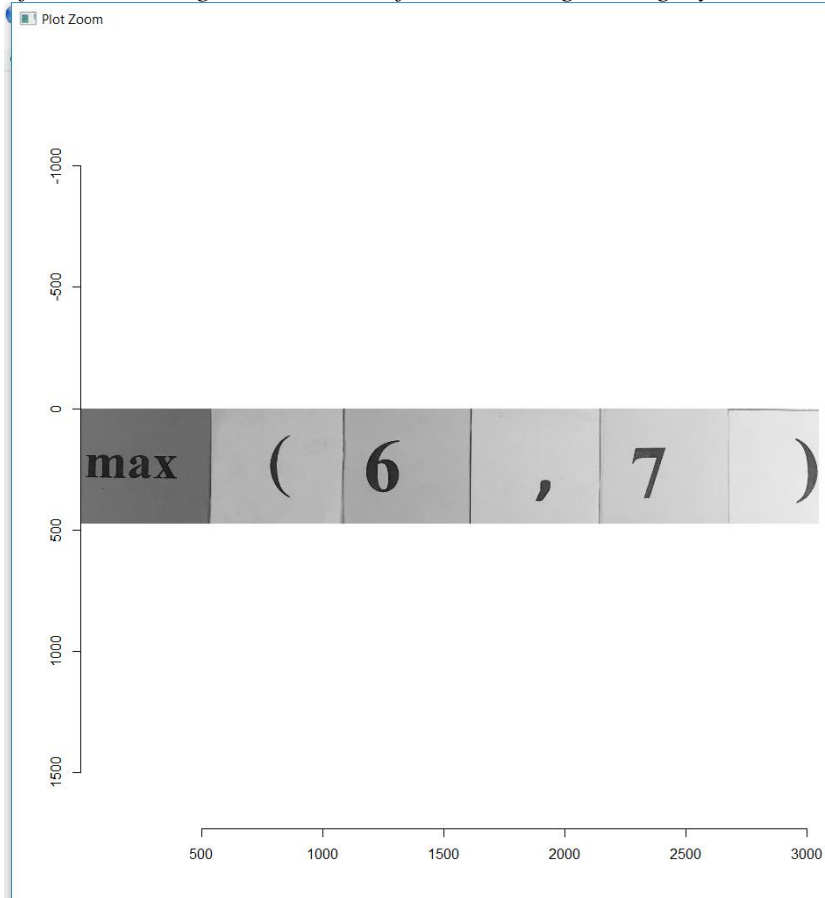


Figure A.19

After converting the maximum function greyscale image into data frame

```
Console Terminal x
~/
> im.g <- grayscale(im)
> plot(im.g)
> im.g <- grayscale(im)
> plot(im.g)
> df <- as.data.frame(im.g)
> head(df,5)
  x y  value
1 1 1 0.4596471
2 2 1 0.4518039
3 3 1 0.4518039
4 4 1 0.4596471
5 5 1 0.4674902
```

Figure A.20

Applying linear model on the data frame of maximum function greyscale image

```
> m <- lm(value ~ x + y, data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.68764 -0.01149  0.00763  0.04675  0.24193

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.816e-01  2.371e-04 2031.20  <2e-16 ***
x             1.401e-04  1.018e-07 1376.62  <2e-16 ***
y            -8.572e-06  6.515e-07  -13.16  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1078 on 1449893 degrees of freedom
Multiple R-squared:  0.5666,    Adjusted R-squared:  0.5666
F-statistic: 9.476e+05 on 2 and 1449893 DF,  p-value: < 2.2e-16
```

Figure A.21

*Before and after trend removal from maximum function greyscale image*

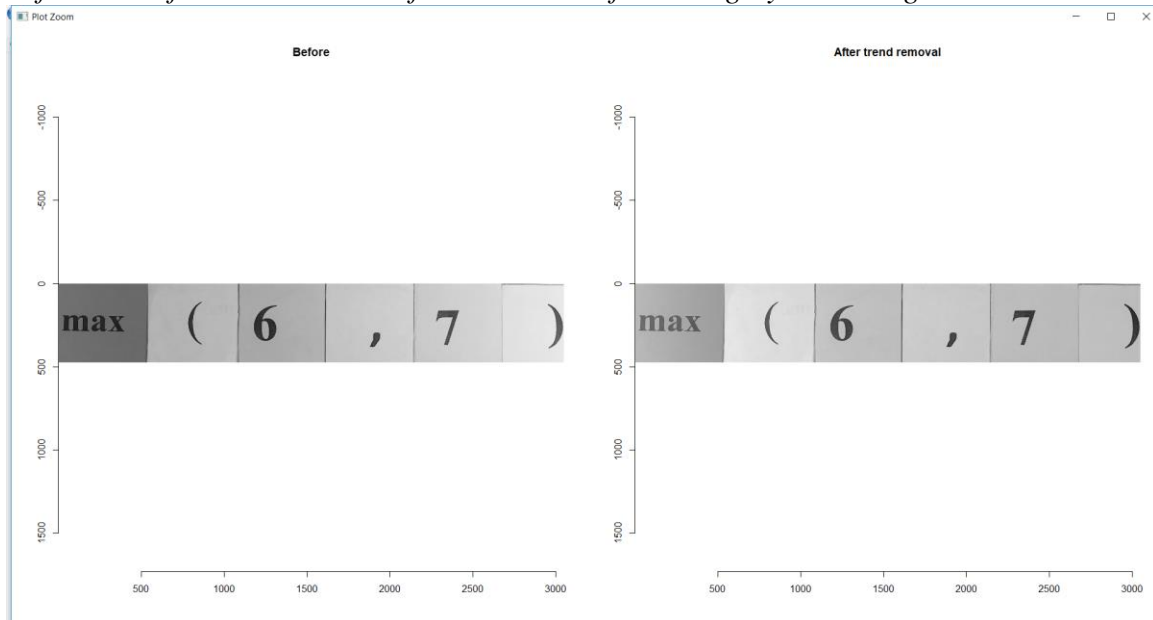


Figure A.22

*Before and after shrink, then grow pixels from maximum function greyscale image*

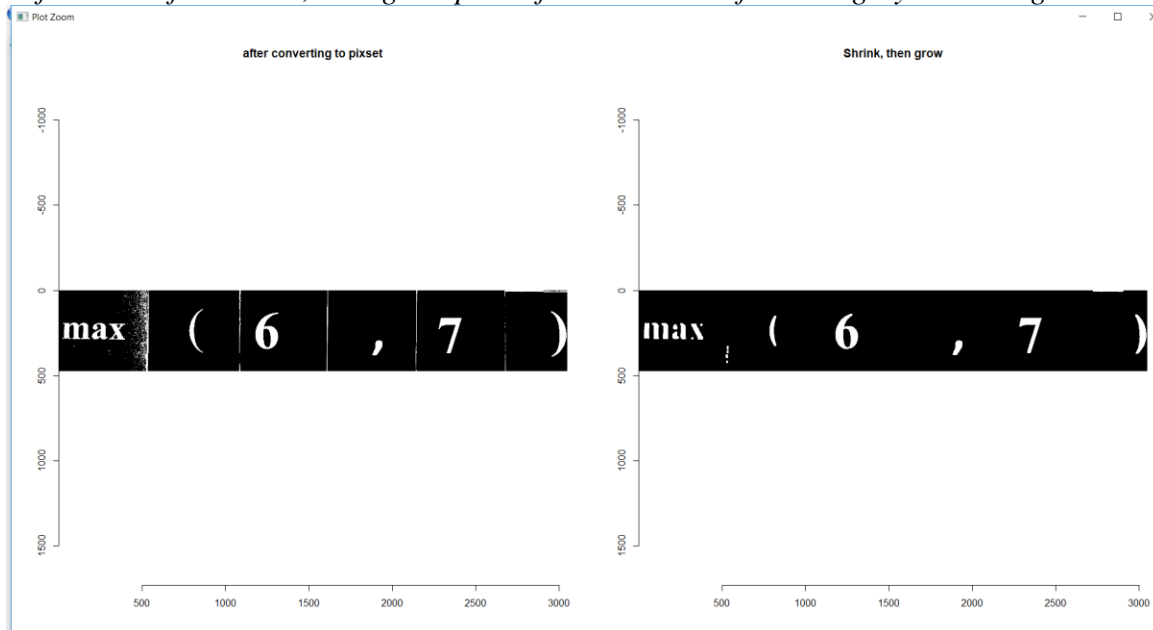
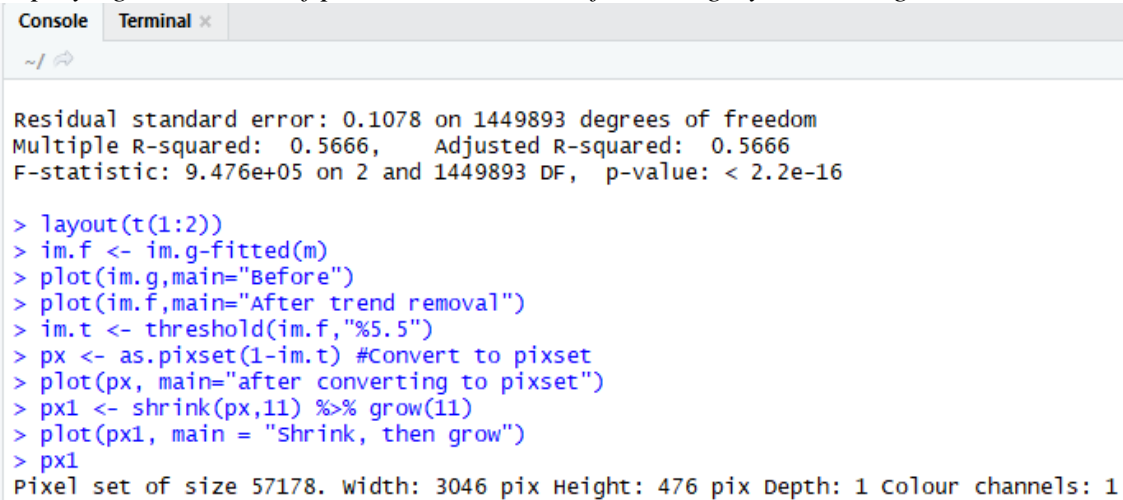


Figure A.23

*Displaying the number of pixels in a maximum function grey scale image*



The screenshot shows a R console window with the following content:

```
Residual standard error: 0.1078 on 1449893 degrees of freedom
Multiple R-squared:  0.5666,    Adjusted R-squared:  0.5666 
F-statistic: 9.476e+05 on 2 and 1449893 DF,  p-value: < 2.2e-16

> layout(t(1:2))
> im.f <- im.g-fitted(m)
> plot(im.g,main="Before")
> plot(im.f,main="After trend removal")
> im.t <- threshold(im.f,"%5.5")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,11) %>% grow(11)
> plot(px1, main = "Shrink, then grow")
> px1
Pixel set of size 57178. width: 3046 pix Height: 476 pix Depth: 1 Colour channels: 1
```

Figure A.24

*Extracting and evaluating text from maximum function greyscale image*

```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
Warning: Invalid resolution 0 dpi. Using 70 instead.
> data1 <- str_replace_all(data," ","")
> cat(data1)
max(6,7)

> eval(parse(text=data1))
[1] 7
>
```

## Log Function

Figure A.25

*A colored image which contains log function*

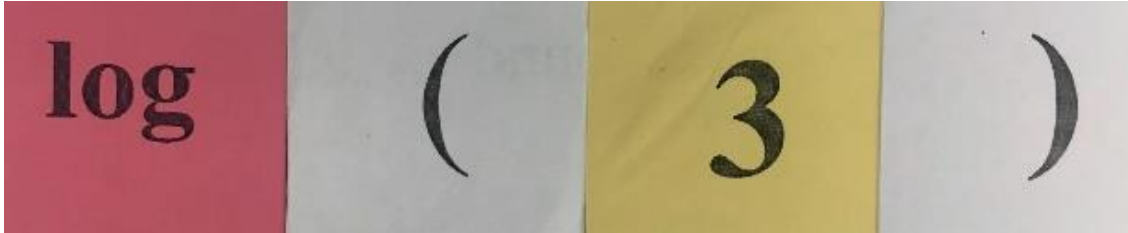


Figure A.26

After converting the log function colored image into greyscale image

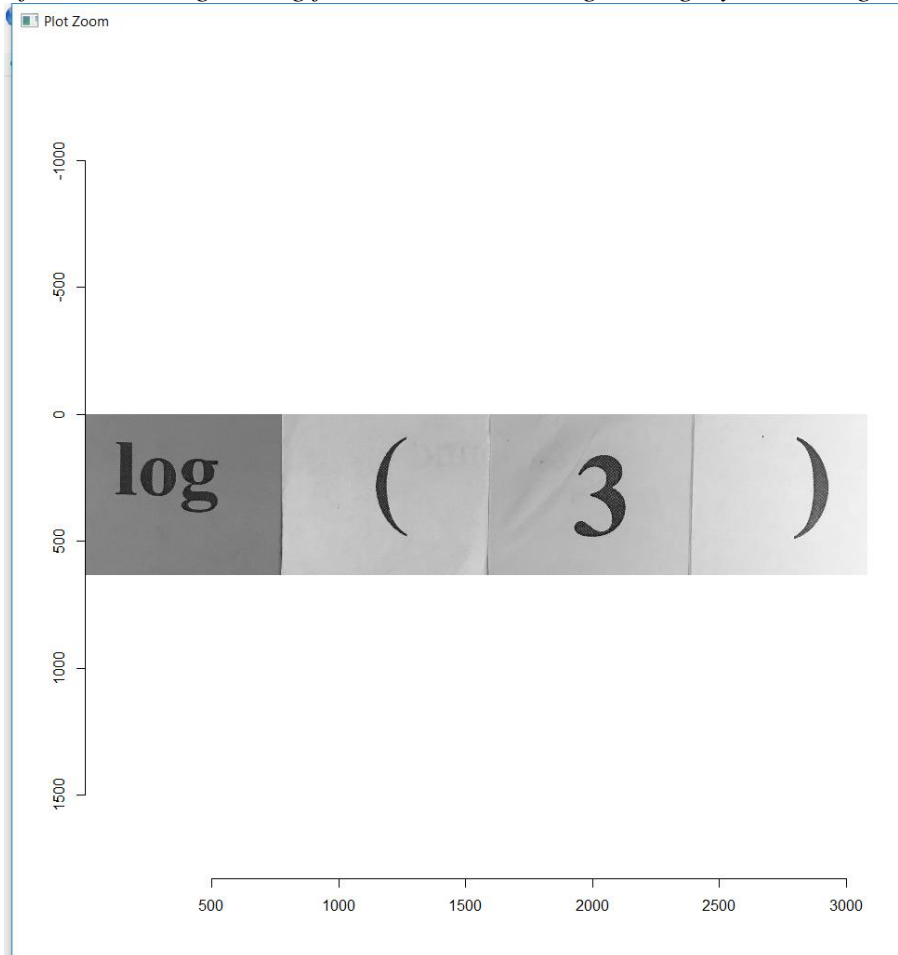


Figure A.27

After converting the log function greyscale image into data frame

```
Console Terminal x
~/
> plot(im.g)
> df <- as.data.frame(im.g)
> head(df,5)
  x y value
1 1 1 0.4481569
2 2 1 0.4442353
3 3 1 0.4442353
4 4 1 0.4481569
5 5 1 0.4520784
.
```

Figure A.28

Applying linear model on the data frame of log function greyscale image

```
> m <- lm(value ~ x + y, data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.68192 -0.03050  0.01076  0.04955  0.22918

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.129e-01  2.036e-04  2028.2  <2e-16 ***
x             1.237e-04  8.642e-08  1431.7  <2e-16 ***
y             5.242e-05  4.201e-07   124.8  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1075 on 1953985 degrees of freedom
Multiple R-squared:  0.5138,    Adjusted R-squared:  0.5138
F-statistic: 1.033e+06 on 2 and 1953985 DF,  p-value: < 2.2e-16
```

Figure A.29

*Before and after trend removal from log function greyscale image*

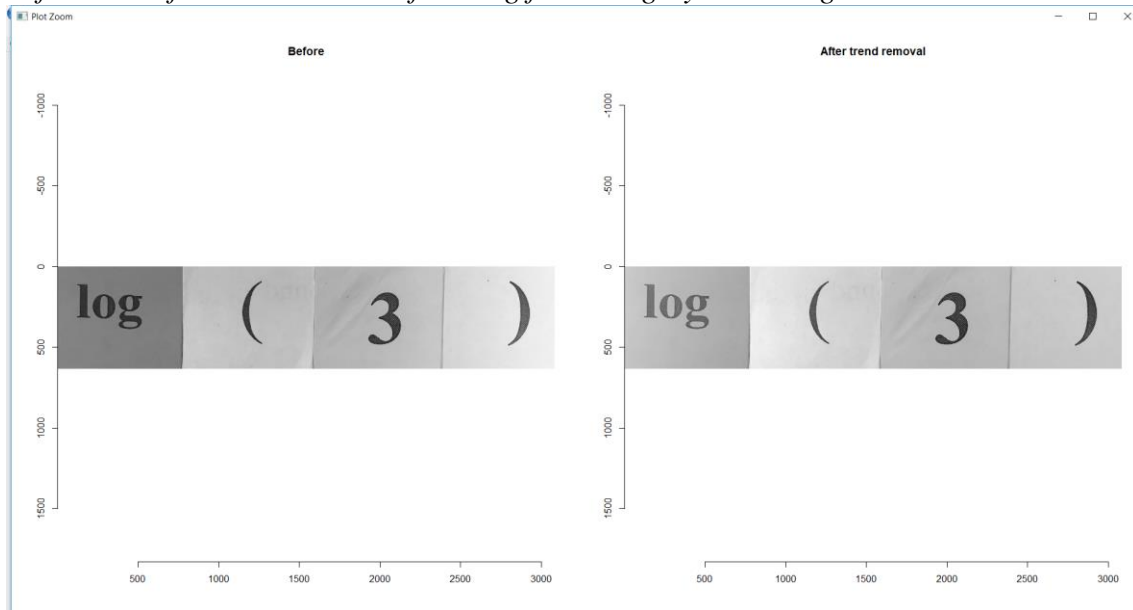


Figure A.30

*Before and after shrink, then grow pixels from log function greyscale image*

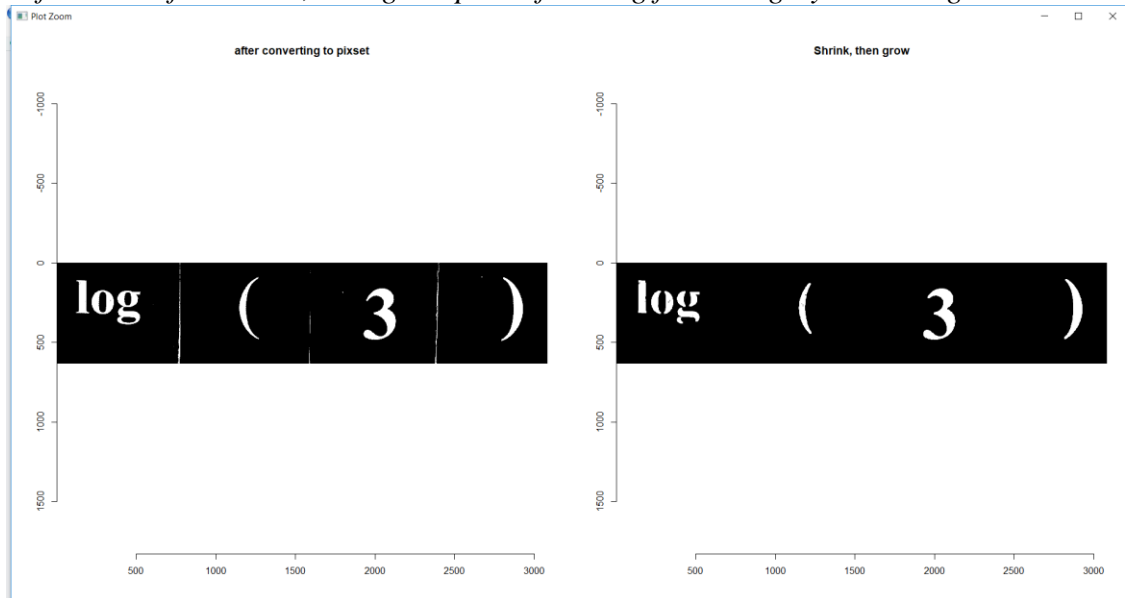


Figure A.31

*Displaying the number of pixels in a log function greyscale image*

```
> plot(px1, main = "Shrink, then grow")
> im.t <- threshold(im.f,"%5.5")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,12) %>% grow(12)
> plot(px1, main = "Shrink, then grow")
> px1
Pixel set of size 88203. width: 3082 pix Height: 634 pix Depth: 1 Colour channels: 1
```

Figure A.32

*Extracting and evaluating from log function greyscale image*

```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
Warning: Invalid resolution 0 dpi. Using 70 instead.
> data1 <- str_replace_all(data," ","")
> cat(data1)
log(3)

> eval(parse(text=data1))
[1] 1.098612
```

## Round Function

Figure A.33

*A colored image which contains round function*

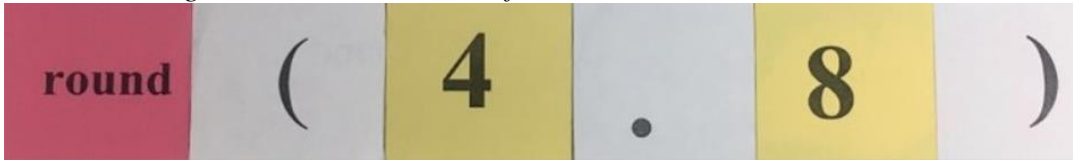


Figure A.34

*After converting the round function colored image into greyscale image*

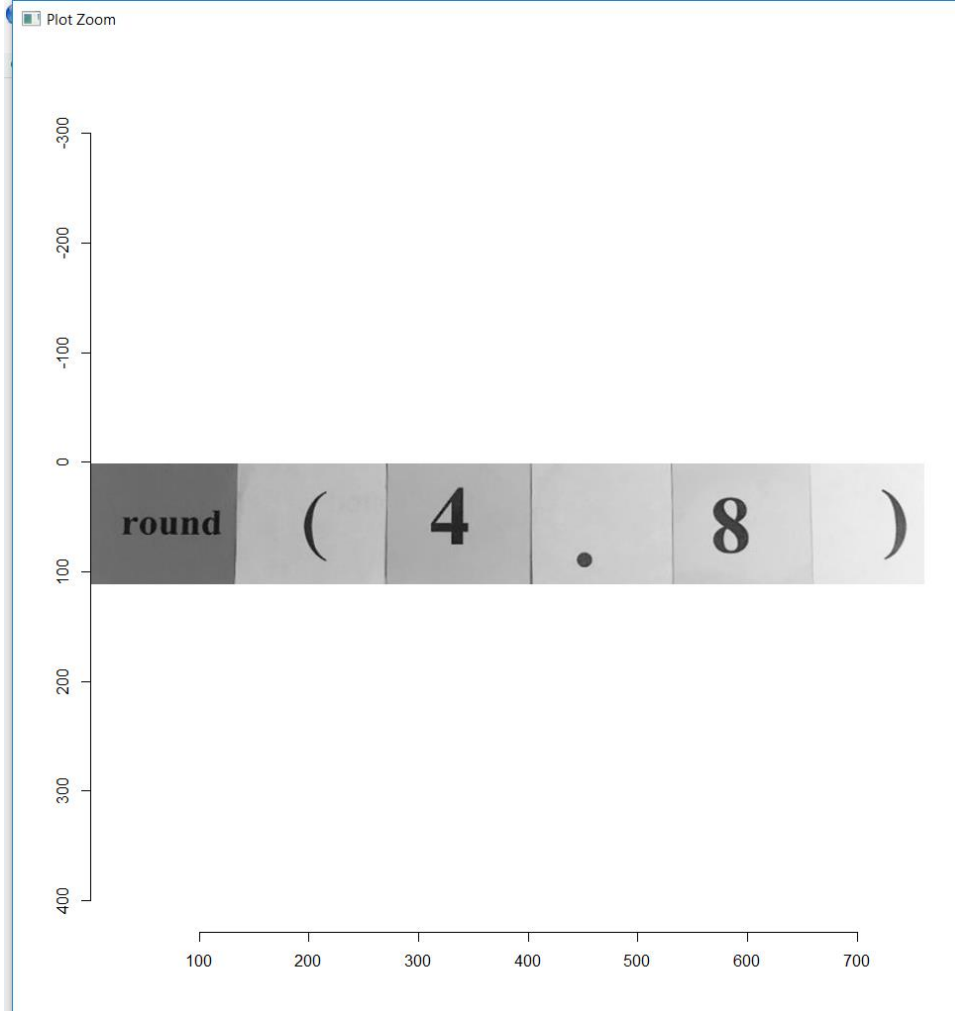


Figure A.35

After converting the round function greyscale image into data frame

```
Console Terminal x
~/
> display(im,rescale = TRUE)
> im.g <- grayscale(im)
> plot(im.g)
> plot(im.g)
> df <- as.data.frame(im.g)
> head(df,5)
  x y  value
1 1 1 0.4521569
2 2 1 0.4521569
3 3 1 0.4521569
4 4 1 0.4510196
5 5 1 0.4510196
```

Figure A.36

Applying linear model on the data frame of round function greyscale image

```
> m <- lm(value ~ x + y,data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.64760 -0.01765  0.01536  0.04370  0.19167

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.876e-01  1.027e-03  474.961  <2e-16 ***
x             5.512e-04  1.759e-06  313.419  <2e-16 ***
y            -1.131e-05  1.206e-05   -0.938    0.348
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1123 on 84468 degrees of freedom
Multiple R-squared:  0.5377,    Adjusted R-squared:  0.5377
F-statistic: 4.912e+04 on 2 and 84468 DF,  p-value: < 2.2e-16
```

Figure A.37

*Before and after trend removal from round function greyscale image*

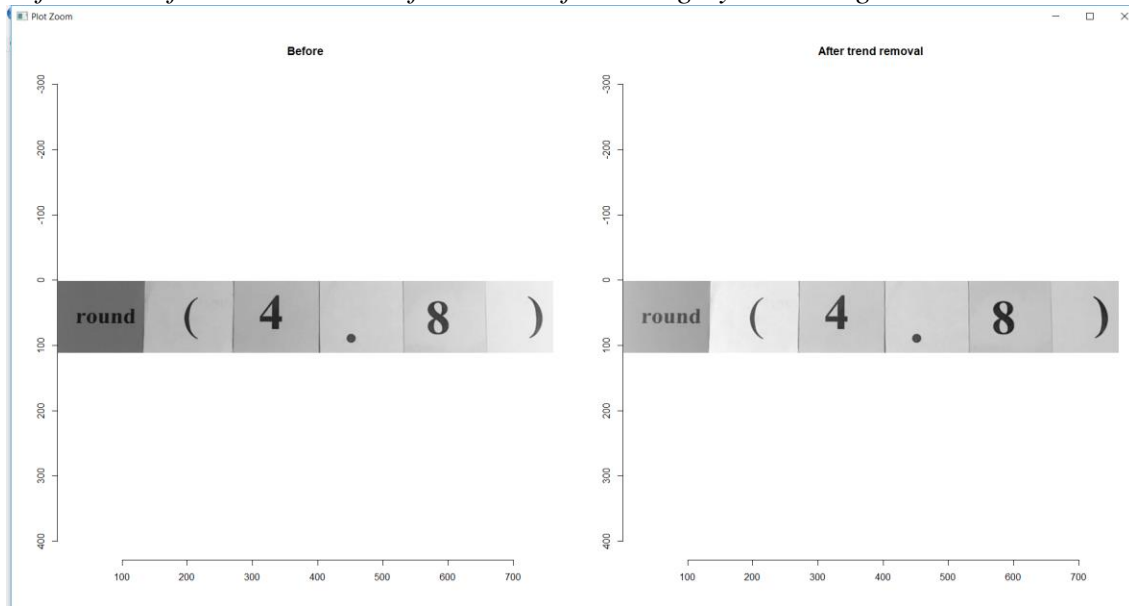


Figure A.38

*Before and after shrink, then grow pixels from round function greyscale image*

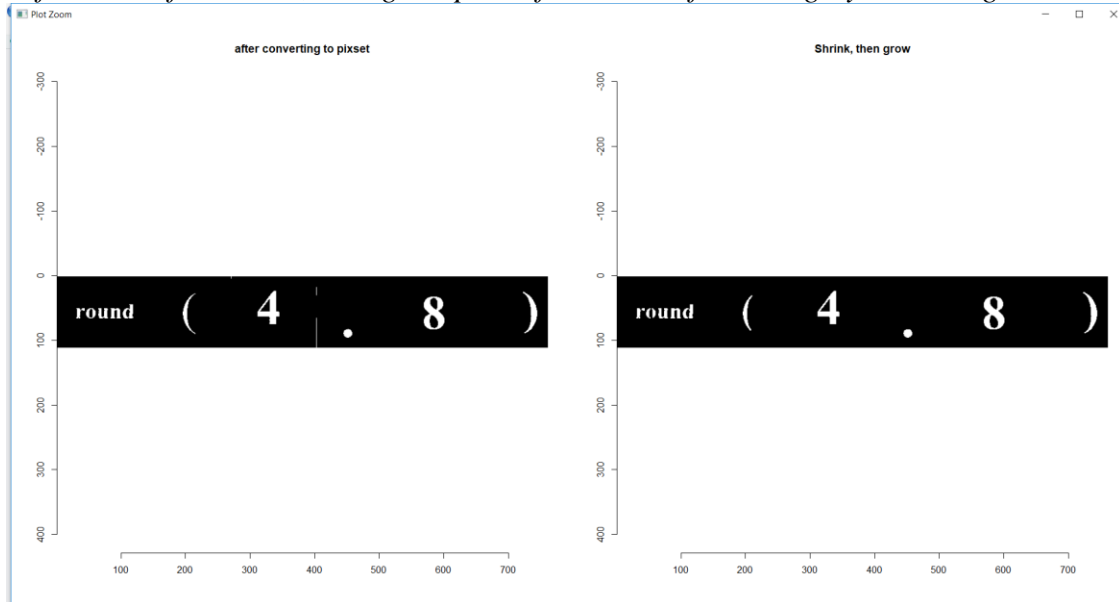


Figure A.39

*Displaying the number of pixels in a round function grey scale image*

```
> im.f <- im.g-fitted(m)
> plot(im.g,main="Before")
> plot(im.f,main="After trend removal")
> im.t <- threshold(im.f,"%11.40")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,2) %>% grow(2)
> plot(px1, main = "shrink, then grow")
> px1
Pixel set of size 3348. width: 761 pix Height: 111 pix Depth: 1 colour channels: 1
```

Figure A.40

*Extracting and evaluating text from round function greyscale image*

```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
warning. Invalid resolution 0 dpi. Using 70 instead.
> data1 <- str_replace_all(data, " ", "")
> cat(data1)
round(4.8)

> eval(parse(text=data1))
[1] 5
```

## Mean Function

Figure A.41

*A colored image which contains mean function*

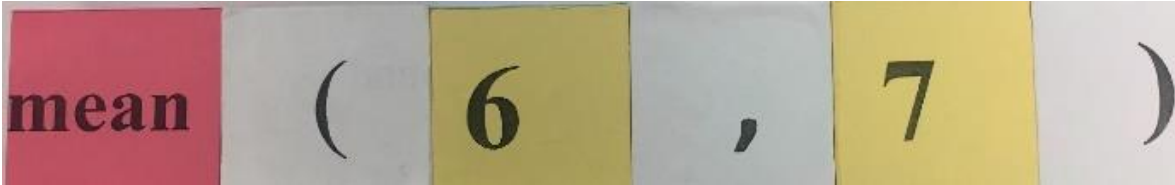


Figure A.42

*After converting the mean function colored image into greyscale image*

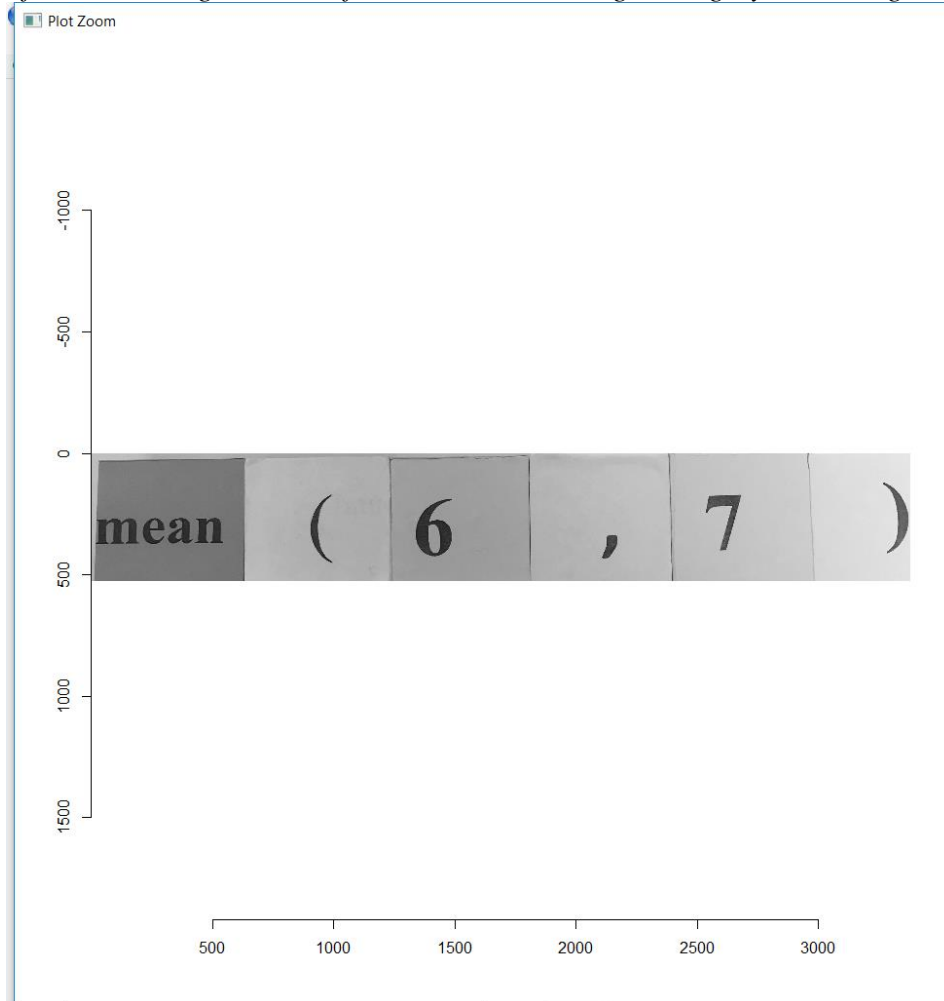


Figure A.43

After converting the mean function greyscale image into data frame

```
Console Terminal x
~/
> display(im, rescale = TRUE)
> im.g <- grayscale(im)
> plot(im.g)
> plot(im.g)
> df <- as.data.frame(im.g)
> head(df, 5)
  x y  value
1 1 1 0.7356863
2 2 1 0.7317647
3 3 1 0.7239216
4 4 1 0.7239216
5 5 1 0.7278431
```

Figure A.44

Applying linear model on the data frame of mean function greyscale image

```
> m <- lm(value ~ x + y, data=df)
> summary(m)

Call:
lm(formula = value ~ x + y, data = df)

Residuals:
    Min       1Q   Median       3Q      Max
-0.66208 -0.02807  0.00648  0.03852  0.32189

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.895e-01  2.085e-04  2347.9  <2e-16 ***
x             1.084e-04  8.077e-08  1341.6  <2e-16 ***
y            -8.294e-05  5.174e-07  -160.3  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.105 on 1779149 degrees of freedom
Multiple R-squared:  0.5064,    Adjusted R-squared:  0.5064
F-statistic: 9.128e+05 on 2 and 1779149 DF,  p-value: < 2.2e-16
```

Figure A.45

*Before and after trend removal from mean function greyscale image*

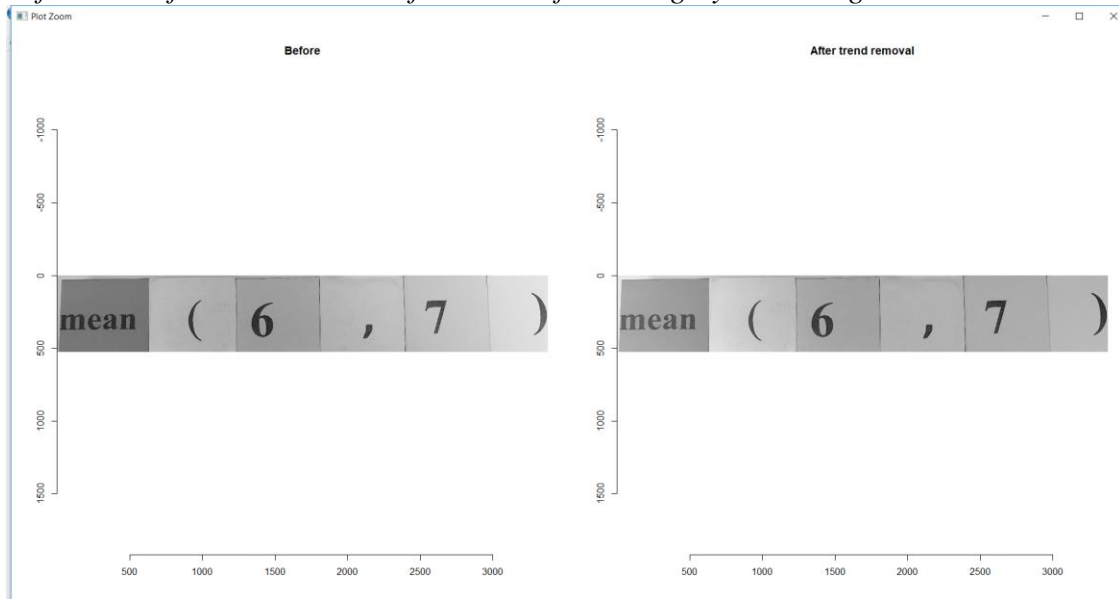


Figure A.46

*Before and after shrink, then grow pixels from mean function greyscale image*

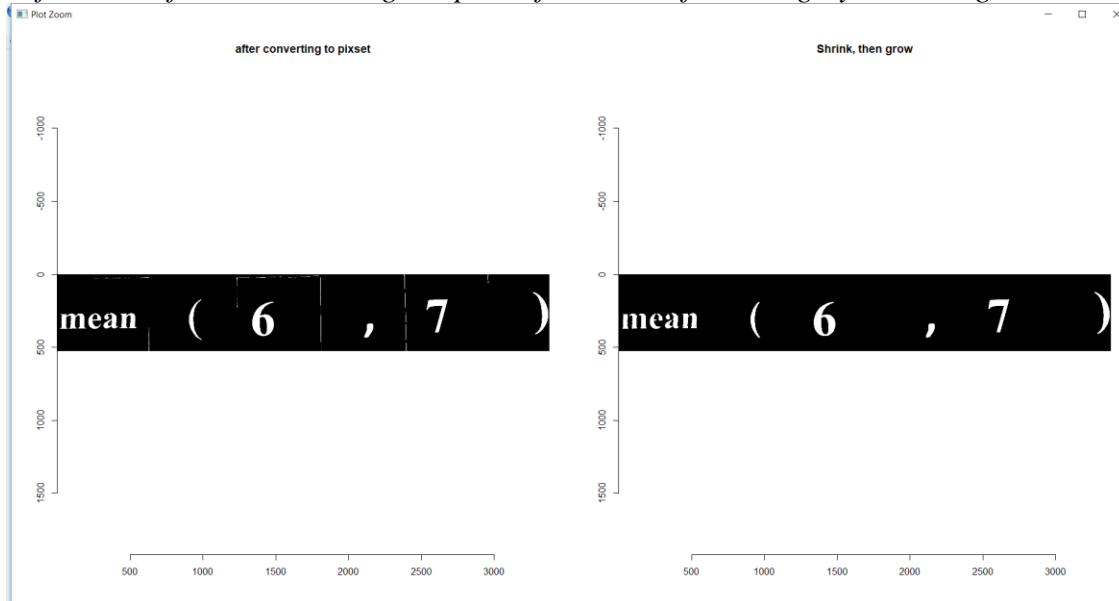


Figure A.47

*Displaying the number of pixels in a mean function grey scale image*

```
> layout(t(1:2))
> im.f <- im.g-fitted(m)
> plot(im.g,main="Before")
> plot(im.f,main="After trend removal")
> im.t <- threshold(im.f,"%4.60")
> px <- as.pixset(1-im.t) #Convert to pixset
> plot(px, main="after converting to pixset")
> px1 <- shrink(px,7) %>% grow(7)
> plot(px1, main = "Shrink, then grow")
> px1
Pixel set of size 73250. width: 3376 pix Height: 527 pix Depth: 1 Colour channels: 1
```

Figure A.48

*Extracting and evaluating text from mean function greyscale image*

```
> impixel <- as.cimg(px1)
> tmpFile <- tempfile(fileext=".png")
> save.image(impixel, tmpFile, quality = .7)
> load.image(tmpFile) %>% plot
> data <- ocr(tmpFile)
warning: Invalid resolution 0 dpi. Using 70 instead.
> data1 <- str_replace_all(data," ","")
> cat(data1)
mean(6,7)

> eval(parse(text=data1))
[1] 6
```