Software Engineering Project Report

Gourab Ghosh Roy, Raabid Hussain and Mohammad Koujan

January 10, 2016

1 Abstract

This project report summarizes the work done by our group for the Software Engineering Project on Seeded Image Segmentation. The paper that motivated all of this work and the method that is implemented using concepts learned in the Software Engineering class is [1]. Our code is available on GitHub at SeededImageSegmentation .

2 Introduction

Segmentation of images is a very important and popular problem in the field of computer vision. It can have applications in several fields ranging from robotics where a robot needs to decide which is the object in a scene to medical imaging where the region of interest needs to be separated out from the background, just to mention a few. One particular subclass of such segmentation problems is the case of seeded image segmentation, when some pixels in the images are labeled. There exist many different methods to move the given labels across the image, like in [2], [3] and [4]. In [5] it was pointed out that most seeded segmentation algorithms are variations of some basic methods like Graph Cuts [6], Random Walker [7] and Watersheds [8]. A major problem of many such algorithms is the computational complexity involved.

The paper [1] used for this project proposes the method of Laplacian Coordinates for seeded image segmentation. The concept of Laplacian Coordinates had also been used in the field of computer graphics as in [9] and [10]. This method uses the minimization of a newly defined quadratic energy functional. As proposed, the method is guaranteed to have a unique solution. Such a solution to the minimization problem is obtained by solving a system of constrained linear equations, which gives the method the major benefit of being easy to implement and suitable for our Software Engineering project.

3 Algorithm

In this section, a brief mathematical description of the algorithm is given, which is based on our understanding of [1]. Most of the notations used are similar for ease of explanation.
Let I denote the image with I_i being the intensity value at pixel i (for an RGB image it is a vector). $G = (V, E, W_E)$ is a directed graph defined where V is the set of nodes representing each pixel in I, E is the set of graph edges consisting of pixels connected in a N-neighborhood (N usually 4 or 8), and W_E is the set of the weights of all such edges. Such a weight is defined as

$$w_{ij} = e^{(-\beta\|(I_i - I_j)\|_\infty^2/\sigma)}, \ \sigma = \max_{(P_i,P_j)\in E} \| \ (I_i - I_j) \ \|_\infty \tag{1}$$

β is a parameter which can be tuned to improve segmentation results. Also as originally suggested in [4], a constant $\epsilon = 10^{-6}$ is added to equation (1) to prevent zero weight values.

The energy functional to be minimized is defined as

$$E(x) = k_1 \sum_{i\in B} \| \ x_i - x_B \ \|_2^2 + k_2 \sum_{i\in F} \| \ x_i - x_F \ \|_2^2 + k_3 \sum_{i\in V} \| \ d_i x_i - \sum_{j\in N(i)} w_{ij} x_j \ \|_2^2$$
$$N(i) = j : (P_i, P_j) \in E), \quad d_i = sum_{j\in N(i)} w_{ij} \tag{2}$$

where k_1, k_2 and k_3 are positive valued constants and B and F are image pixels seeded as background and foreground respectively.

The last term in (2) which measures how much a pixel varies in comparison to the weighted average of the pixels in its neighborhood can be written in terms of the graph Laplacian matrix L as

$$\sum_{i\in V} \| d_i x_i - \sum_{j\in N(i)} w_{ij} x_j \|_2^2 = \| Lx \|_2^2$$

$$D_{ij} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad W_{ij} = \begin{cases} w_{ij} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$L = D - W$$

Setting the k_i s to 1 , the equation (2) can be expressed as

$$E(x) = x^t(I_s + L^2)x - 2x^t b + c$$

$$b(i) = \begin{cases} x_B & \text{if } i \in B \\ x_F & \text{if } i \in F \\ 0 & \text{otherwise} \end{cases} \quad I_s(i,i) = \begin{cases} 1 & \text{if } i \in S = B \cup F \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where c is a constant.
As pointed out in [1], $(I_s + L^2)$ matrix is symmetric and positive definite. The minimum is given by the solution of the following linear system [11]

$$(I_s + L^2)x \; = \; b \tag{5}$$

After obtaining the solution x from equation (5), the segmentation label is finalized by

$$y_i = \begin{cases} x_B & \text{if } x_i \geq (x_B \; + \; x_F)/2 \\ x_F & \text{otherwise} \end{cases} \tag{6}$$

For multiple region segmentation, the equation (5) becomes N system of linear equations as

$$(I_s + L^2)x^{(j)} \; = \; b^{(j)} \tag{7}$$

where $b^{(j)}$ is different for the given N labels. Segmentation labels are obtained by

$$y^{(j)} \; = \; \cap_{\substack{p=1,2\ldots N \\ (p\neq j)}} (x^{(j)} > x^{(p)}) \tag{8}$$

4 Implementation

This section highlights the key features in our implementation of the seeded image segmentation algorithm. We used C++ as the programming language and Qt as the platform. Since the project is dealing with images, OpenCV [12] library was quite an obvious choice. Also there are extensive matrix operations and solving of linear systems involved, so Eigen library [13] was used for the linear algebra part.

4.1 Efficient Use of Eigen Library

Our code uses the SparseMatrix class [14] available in the Eigen library. First Mat datatypes [16] in OpenCV library were used to store the images. Then we used similar Mat objects to store the Weight matrix W as defined in equation (3) in the previous section. Now for a $M * N$ size image, the size of such a matrix is $M^2 * N^2$. When the first version of our code was tested on a $500 * 500$ image, the program crashed because a Mat object was unable to store those many values. This motivated the need of using a new class. We would like to comment on the clarity of the paper [1] which points out the inherent sparsity of the associated matrices. Considering all these aspects, SparseMatrix Class was quite a good choice.
For solving linear system of equations when the coefficient matrix is a SparseMatrix, there are many options available in the Eigen Library. The SimplicialLDLT class of the SparseCholesky Module [15] is used for our case. We also tested with SimplicialLLT of the same module to investigate whether any improvement in runtime or segmentation results would be obtained, but the results were exactly identical as in LDLT.

4.2 Object Oriented Programming

In this section, our main focus is on describing the OOP aspect of our program. Since our program is written in C++ language, which is an OOP language, our aim was to fully exploit the advantages of this language in terms of modularity, reusability, and readability. Another goal was to provide a good data structure that is useful and flexible to use for the Graphical user interface (GUI) we have.
Initially we started writing our program in a procedural way which would produce a simple but dirty solution, by implementing everything under the

main function. This was in order to be familiar with the general data structures, methods and the libraries that we needed. Once this initial part was done, our next goal was to move to the OOP way of implementation. The idea in this method of seeded image segmentation, as stated in [1] , is mainly about solving a system of linear equations of the form $Ax = b$. Choosing to encapsulate every parameter that is a part of this system of linear equations and defining the required methods to manipulate those parameters is a fundamental design issue. Therefore, we defined the required parameters stated in equations 1 and 4 as private data members of the class we wrote "SeededImgSeg", and implemented the necessary class methods that would help us to set and get those parameters in an efficient and controlled way.
For the mathematical calculations we separated the steps that are essential to solve the linear system into three main methods : computing the seeds-dependent matrices (I_s,b), computing the seeds-independent matrices (D,W) which are required to find the Laplacian matrix L, and then segmenting the image by solving the symmetric, positive definite and sparse linear system.
We implemented a number of necessary functions like constructors (default and parameterized), destructor, and an overloading assignment operator which facilitates the manipulation of objects created from our class. This method of implementation, in fact, has a high impact on our program in terms of easing the process of implementing the multi-region segmentation part of our program. We just had to make some modifications to the seeds-independent and dependent functions, keeping the function of calculating the Laplacian matrix intact. Moreover, this helped us considerably in fixing bugs, modifying the different methods, avoiding code duplication, improving readability and in linking this part of our code to the GUI classes part.

4.3 Graphical User Interface

In order to make our program user-friendly, a graphical user interface (GUI) was designed. Our code uses the Qt Widgets Application provided by Qt to design and generate the user interface. Since the application deals mainly with images and user inputs, using this library seemed a good choice. Since none of us had any prior knowledge and experience of creating GUIs in Qt, online tutorials by [24] and [25], Qt documentation [26] and [27] were consulted.
The application starts with a main form that asks the user to enter the number of segmented regions he or she desires. This form uses the radio buttons

for each of the choices the user has. In our case, we restricted the maximum number of segmented regions to be four. This was just chosen to be a reasonable limit depending on the nature of the task, however this limit can be easily increased by simply including checkboxes for higher levels of segmentations. In order to restrict the user from selecting more than one type or level of segmentation, all the radio buttons were placed inside a group box. The first user interface that appears after running the program is depicted in the following figure.

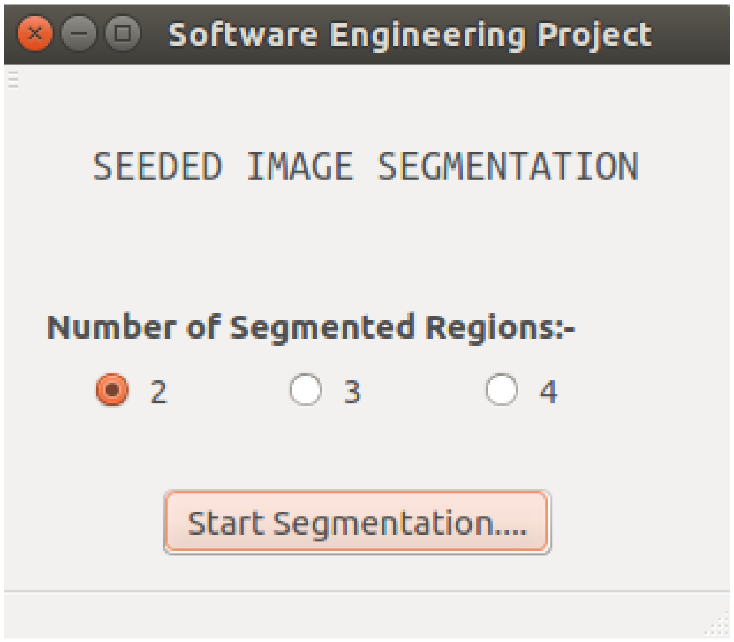
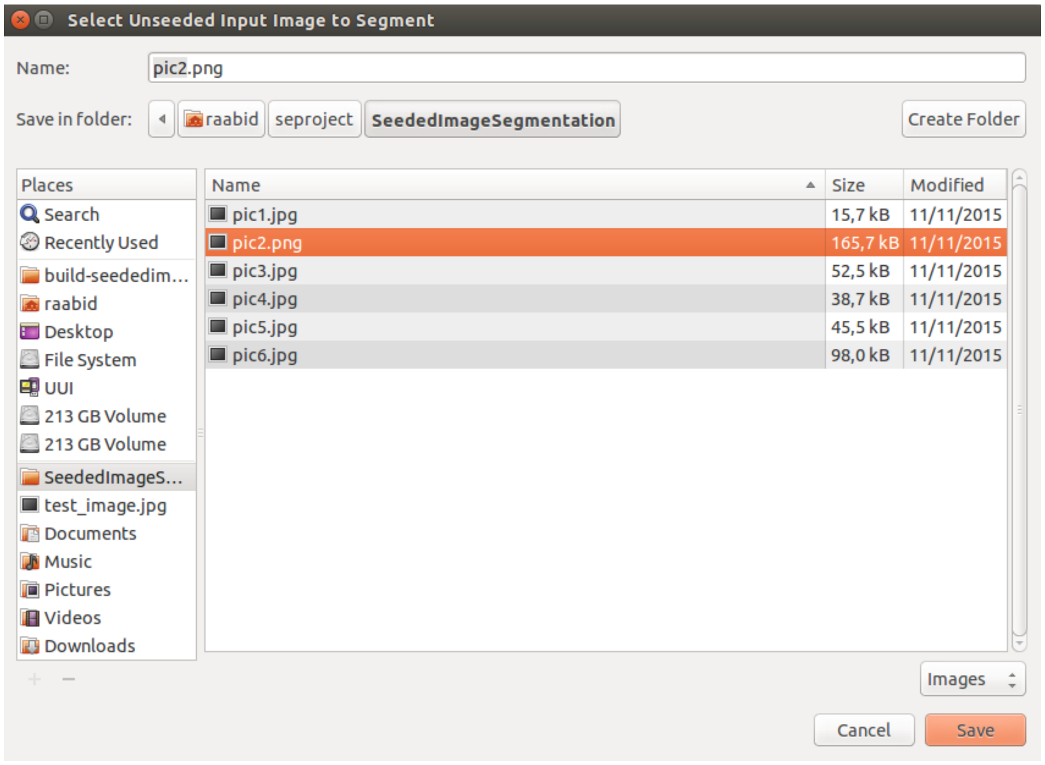

Software Engineering Project
SEEDED IMAGE SEGMENTATION
Number of Segmented Regions:-
2
3
4
Start Segmentation....

Figure 1: GUI: Selecting the number of segmentations

The two region segmentation has been selected as default in case the user forgets to select the level of segmentation. When the user clicks on the "Start Segmentation...." button, a dialog box opens which asks the user to select the image that needs to be segmented. The image can be chosen from any location on the user's computer. The dialog box is shown in the following figure.



Select Unseeded Input Image to Segment

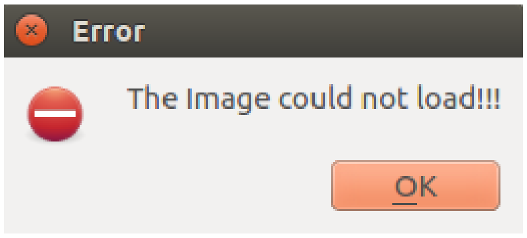
Name: pic2.png

Save in folder: raabid / seproject / SeededImageSegmentation

Name	Size	Modified
pic1.jpg	15,7 kB	11/11/2015
pic2.png	165,7 kB	11/11/2015
pic3.jpg	52,5 kB	11/11/2015
pic4.jpg	38,7 kB	11/11/2015
pic5.jpg	45,5 kB	11/11/2015
pic6.jpg	98,0 kB	11/11/2015

Figure 2: GUI: Selecting the image to be segmented

This dialog box has been designed such that the user can only select image file formats ('.jpg', '.png' and '.bmp'). After the user has selected the image file and confirmed, the software checks whether the selected image is a valid image. If the selected image is empty or can not be read by the software, the software generates an error message.



Error

The Image could not load!!!

Figure 3: GUI: Error message for empty or corrupted input image

If the input image is a valid image then a new user interface appears which displays the input image on a QLabel object and the user can now enter the seeds on the image. This is a user-defined QLabel object inherited from the Qt's Qlabel class. The inherited class is designed to read the mouse location with respect to the image object itself and also to draw on the image using QPainter class. The seeds are input to the image by drawing a point (of 10 pixels diameter) at the location of the mouse only if the mouse is on the image with left button pressed. So, the basic purpose of this additional class is to let the drawing process run at the back-end by isolating it from the front-end.

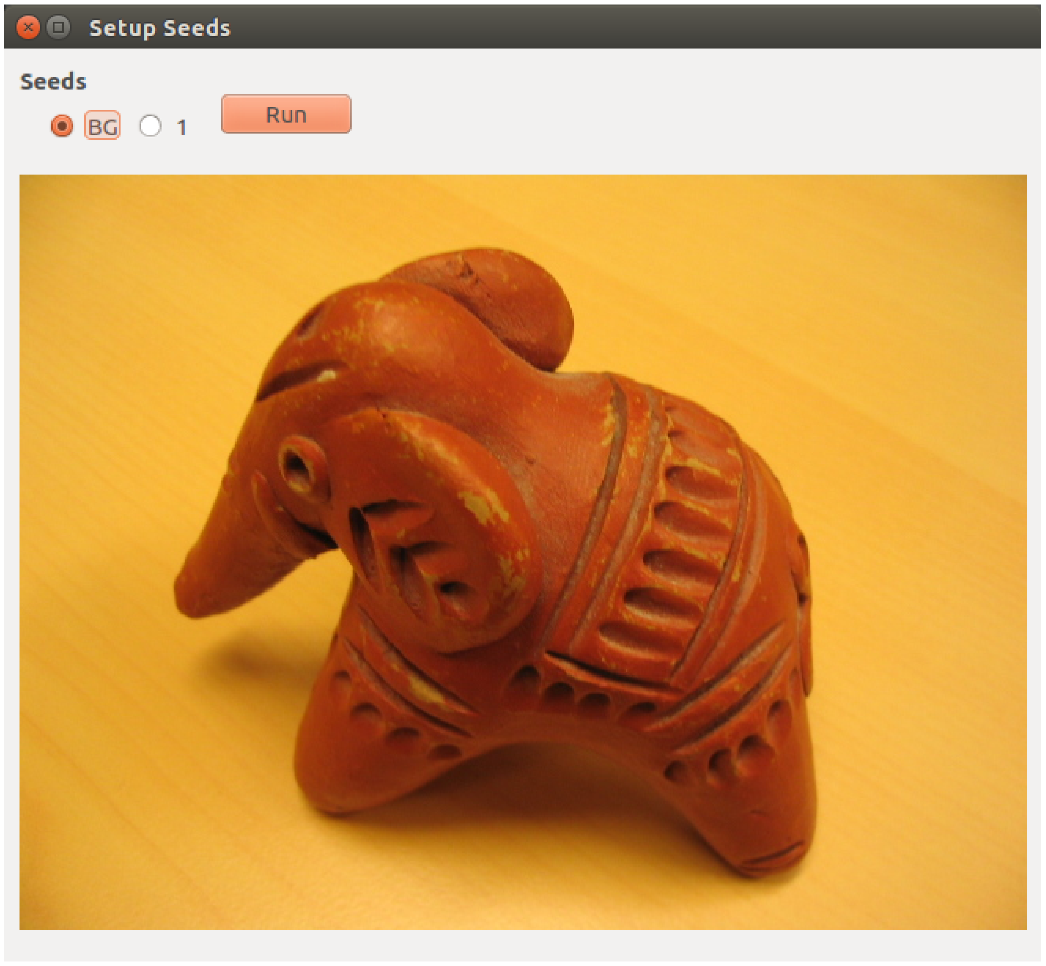


Figure 4: GUI: Entering seeds on the image

The program is designed that the user first selects the type of seed to be inserted from the checkboxes above the image. Only the desired number of seeds (chosen on previous UI) are shown. After selecting the seed, the user can draw the seeds on the image. After drawing all the seeds on the image, the user clicks on the "Run" button to start the segmentation algorithm. Since the segmentation algorithm is complex in terms of computational load, the title of the user interface automatically changes from "Setup Seeds" to "Please wait: The segmentation process is going on!". Once the segmentation process has finished a new user interface appears displaying two images: seeded image and the segmented image as shown in the following figure.

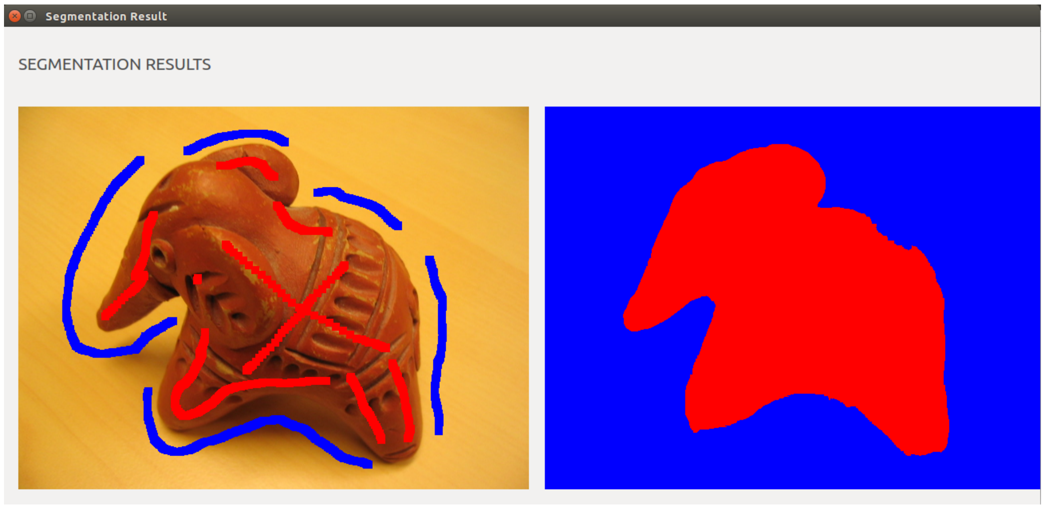


Figure 5: GUI: Segmentation results

Each individual region is shown by the colour used to represent the seed it belongs to.
In order to start a new segmentation, the user closes the form and the first UI asking for the level of segmentation reappears. In order for the program to not encounter any runtime errors and be user-friendly, all the user inputs have been designed in such a way that user only has to select the options rather than manually inserting them.

4.4 CMake and Unit Tests

The code was modified so that it can also be run through CMake. This was done to make the software independent of the platform being used to design it. The advantage of CMake is that it does not require any other software to build the code and it has the ability to automatically link different packages and dependencies required by the program. The CMake version was created using a skeleton available at [28]. Since we used '.ui' files to create the graphical user interface, this skeleton with automoc features in makefiles needed to be used to build the code.
Another key feature of CMake is that it allows programmers to create custom tests to test different features of the software. A unit test was also implemented in our code. It was a simple test that checks the opencv functionalities of the software by reading an image and checking its size, to make sure libraries have been correctly linked and the program is ready for execution through the GUI.

5 Results

This section summarizes the results of our experiments.

5.1 Quantitative Results

In this section we present the results of applying the Laplacian Coordinates (LC) segmentation method on some images selected from the "Grabcut" dataset, which was introduced in [17] . Along with the original images, there are seed images and also the ground truth images. The dataset is obtained from Microsoft Research Cambridge website [18]. It also includes images from the Berkeley Image Segmentation Benchmark Database [19] .

Rand Index (RI) values [20] are computed to quantitatively deduce the effectiveness of the segmentation process. RI is a measure of the similarity between the segmented image and the ground truth in terms of the number of pixel pairs that share the same label, so higher values are desirable. The paper in [1] gives a comparison of quality metrics averaged over the entire dataset. In this project we focus individually on some particular images of the dataset and present RI values separately for each one of them.

The results of LC method are compared with the results of Random Walker Method proposed in [7] . The MATLAB code can be found at [21] . We used the Python implementation [22] which is part of Python scikit-image [23] . The Table **??** shows the results of the comparison on 7 images from the "Grabcut" dataset. For the ease of reproducing the results, the particular images used are also mentioned in the following table.

Image	RI Value for RW	RI Value for LC
Elefant	0.9565	**0.9589**
Memorial	**0.9525**	0.9324
Person2	0.8848	**0.8900**
Person7	0.9459	**0.9817**
Person8	0.9565	**0.9589**
Music	0.9830	**0.9859**
Person3	0.9436	**0.9527**
Average	0.9462	**0.9516**

Table 1: Comparison of RI Values between LC and RW Methods

For better visualization, the same metrics are shown in a graph in Figure 6 .

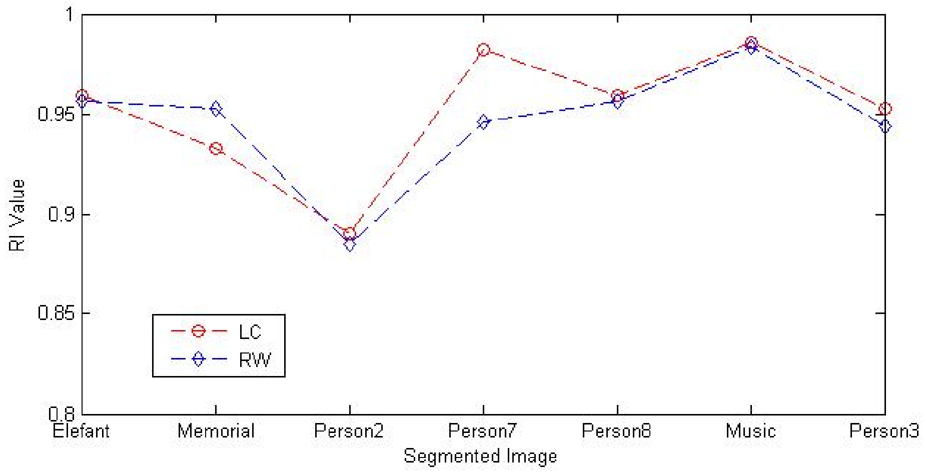


Figure 6: Comparison of RI Values for LC and RW methods

As observed here, LC method outperforms RW in all the images except one, and is better on an average. Such behavior as demonstrated already in [1] is expected, so in our project we reconfirm the findings of the authors. It is understood that the performance of both the LC and RW methods are dependent on the suitable choice of corresponding tuning parameters. In this case we present the best RI values for both methods obtained over different values of those parameters. Effect of such a parameter on LC segmentation will be demonstrated in further details in Section 5.3.

5.2 Qualitative Results

Though the segmentation results have been quantitatively compared in Section 5.1, it is useful to look at the output because often human eye is a better judge of the segmentation result than label similarity values. The images are the same and in the same order of display as mentioned in the previous section. The foreground (object) is represented in white and the background in black for comparison with the given ground truth.

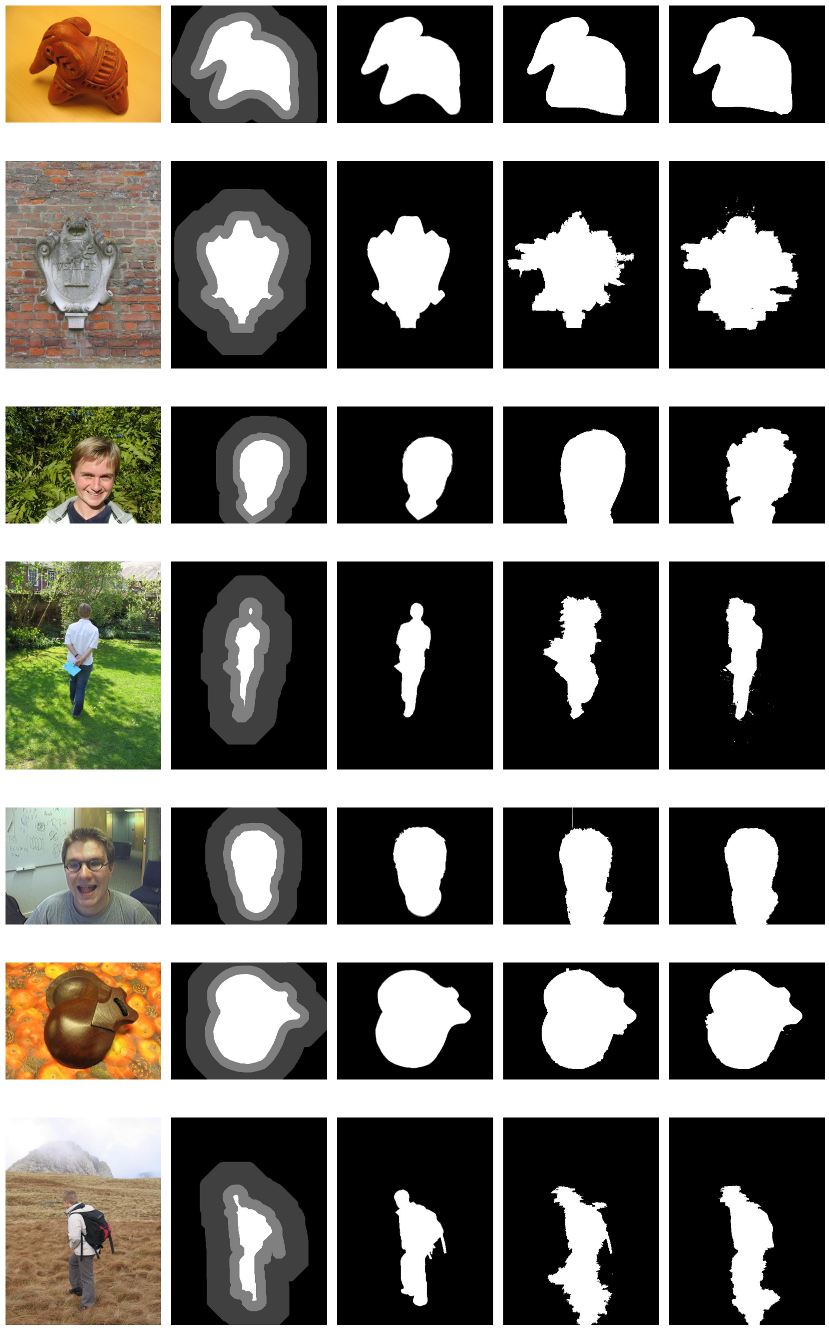


Figure 7: Segmentation Results with LC and RW methods : From left to right - Original Image, Given Seed Map, Ground Truth, RW Output, LC Output

Figure 7 shows that Laplacian Coordinates method produces better segmentation output than RW in all the test images except the memorial image (second from top). Also the boundaries between foreground and background are in general smoother for the segmentation output of LC method. This matches with the behavior the authors of [1] had observed in their tests too.
.

5.3 Effect of Parameters

In this section we present the results of our experiments with different parameters in the algorithm.

5.3.1 Effect of β and σ

The ratio of the tuning variables β and σ as in Equation 1 controls the weight values between pixels and strongly influences the result of the segmentation task. We fixed $\sigma = 0.1$ as in [1] and experimented with different values of β. In Figure 8 the segmentation output of Laplacian Coordinates method on one particular image (Person7) used in earlier sections are shown, for three widely varying values of β , along with the ground truth.

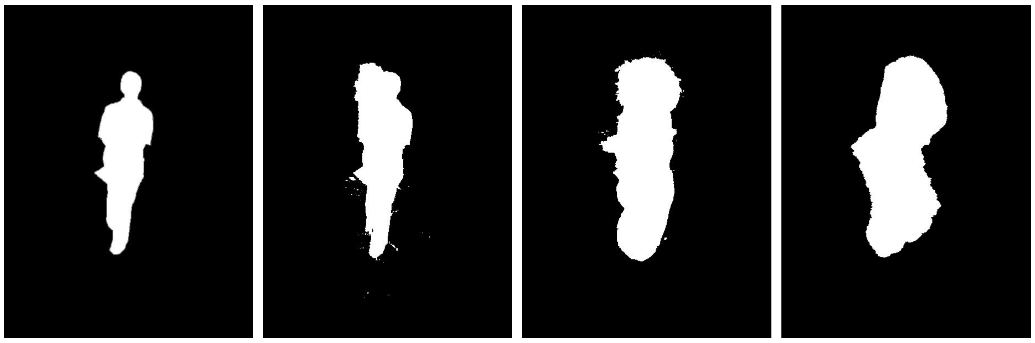


Figure 8: Effect of changing β on LC Segmentation. From left to right - Ground Truth, LC Segmentation with $\beta = 0.0025$, $\beta = 0.00025$, $\beta = 0.025$

The corresponding RI values are given in Table **??** .

The value of β which produced best results for this particular image could produce a poor result in some other image. It can be concluded that selection of the optimal tuning parameter for an image is very important for obtaining good segmentation.

β	RI Value for LC output
0.0025	0.9817
0.00025	0.9386
0.025	0.9230

Table 2: Comparison of RI Values for different β

5.3.2 Effect of Neighborhood size

The authors in [1] mention the use of 4 and 8 neighborhoods for calculating the pixel weights. Our code has the ability to run with both values. Both values were tested in our implementation, and not much difference was found in the segmentation output. The run time of the program for the 8 neighborhood case increased by a marginal fraction, only measurable in milliseconds for large images.

5.4 Execution Times

For real-time applications of image segmentation, the execution time of the algorithm is very important. As expected, the run time increases with the size of the image segmented. To better comprehend the nature of such increase, in Figure 9 we plot the run time of the LC segmentation process versus the number of pixels in the input image. The exact values would depend on the specifications of the machine the code is being run on. This graph has been generated for segmentation run times on an Intel(R) Core(TM) i5-5200U CPU 2.20 GHz with 4 GB RAM. But rather than actual values, the trend here is more informative.

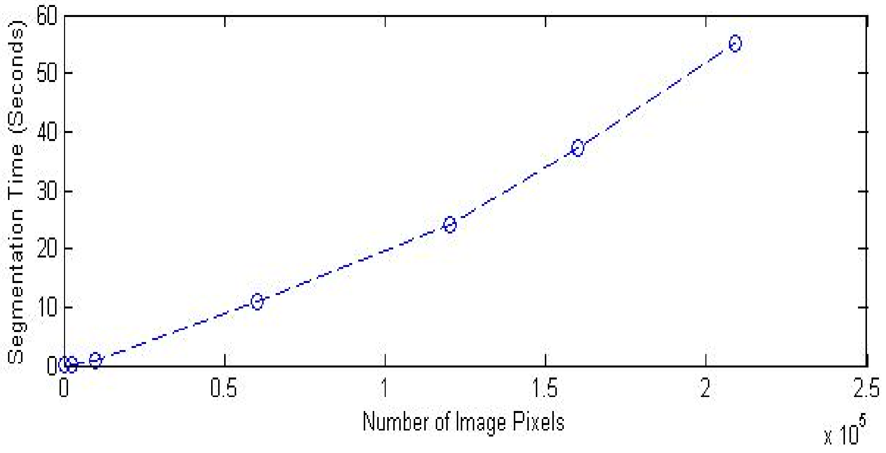


Figure 9: Segmentation Run Time vs Number of Image Pixels

The changes for lower pixel values can be better observed with a run time versus logarithm (base 10) of the number of image pixels, as in the following figure.

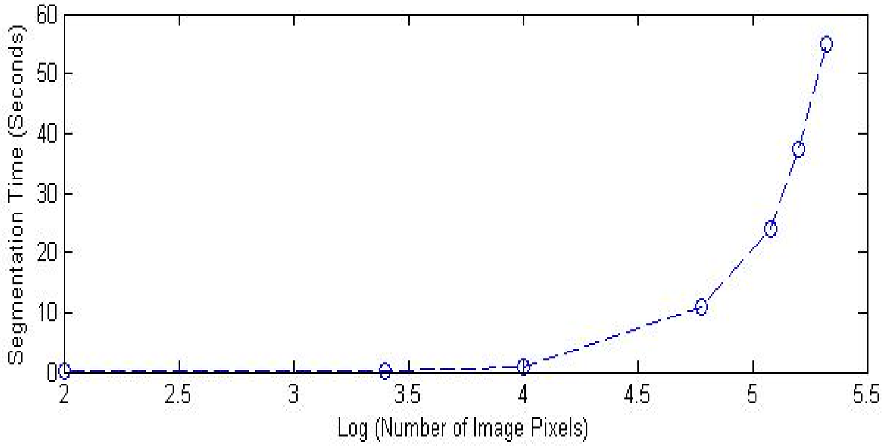


Figure 10: Segmentation Run Time vs Logarithm of Number of Image Pixels

For images with high number of pixel values, the major segment of the execution time is spent by the linear system solver [15]. The code was optimized over several iterations of coding where now the fraction of the time spent by the program on building the weight matrix from the input image is very less. One such optimization was incorporating the use of triplets for filling values into the SparseMatrix objects, as outlined in [14]. This step reduced a considerable amount of execution time.

5.5 Multi Region Segmentation

In order to increase the capabilities of the software, a multi-region segmentation capability was added using the scheme proposed by the authors using (8). In order to achieve this, muliple number of seeds (up to four in our case) were taken as input from the user and segmentation steps were repeated for every seed level. The concept used here is to run the segmentation algorithm by taking one seed as foreground and all the other seeds as background. This process is repeated for all the seeds. The results of each individual segmentation are compared to decide which pixels belongs to which seeded region. The following figure displays the result for such a 3 region segmentation.

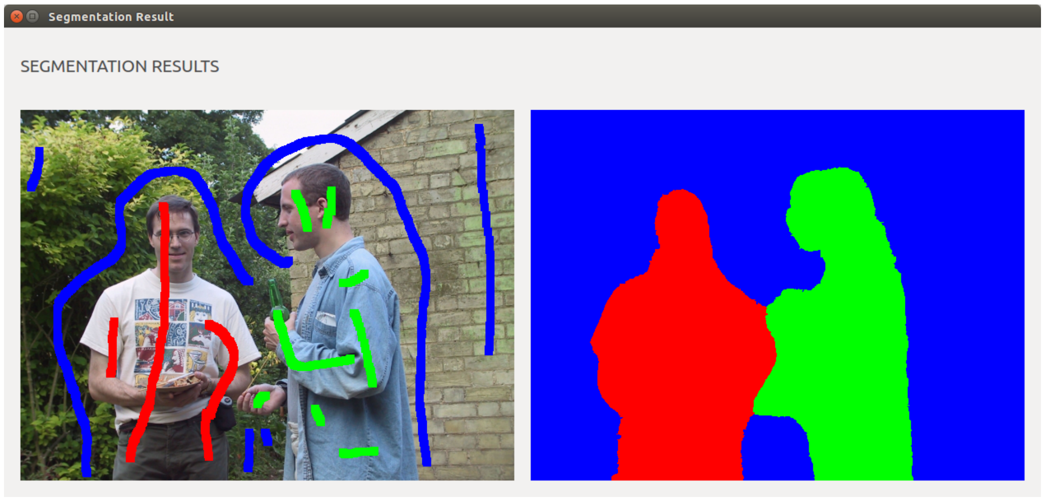


Figure 11: Result of multi-region segmentation

In the initial version that was formulated for multi-region segmentation, the entire segmentation algorithm was re-run to get individual results for each seed (used as foreground). However, this was deemed to be computationally inefficient as the run time for N level segmentation was equated to be N multiple of the time taken for single region segmentation. Since only the b in (7) was different for each individual level, the algorithm just computes the $(Is + L^2)$ matrix in (7) only once and solves the linear system for b according to the level. This helped to reduce the runtime for multiple region segmentation significantly. The comparison of final execution times between number of segmentations are shown in the following figure.

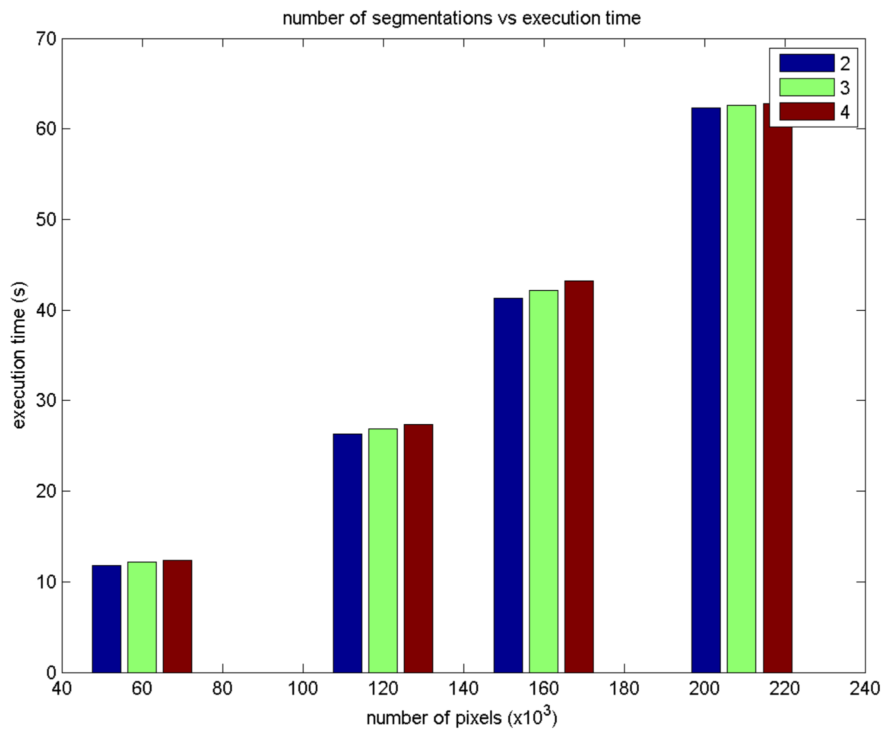


Figure 12: Execution time vs number of seeds

6 Project Management

In this section we presents our work progress in this project. We mainly used "Project" program from Microsoft to organize our work by determining the required tasks and setting a start and end date for each task.

The following table also shows the time specified for each task during our work.

Task Name	Duration (days)	Start Date	End Date
Reading the research paper	10	27/09/2015	08/10/2015
Writing first version	16	10/10/2015	30/10/2015
Modifications and Optimizations	10	01/11/2015	12/11/2015
Testing	4	15/11/2015	18/11/2015
GUI Implementation	10	20/11/2015	03/12/2015
OOP implementation	7	22/11/2015	30/11/2015
GUI Integration	3	01/12/2015	03/12/2015
Multi-region Segmentation	7	05/12/2015	14/12/2015
Further Optimization	5	15/12/2015	21/12/2015
Testing and Comparisons	3	28/12/2015	30/12/2015
CMake Test	3	06/01/2015	08/01/2016
Writing the Report	19	15/12/2015	08/01/2016
Writing the Presentation	2	08/01/2016	10/01/2015

Table 3: Duration of Project tasks

7 Conclusion

The seeded image segmentation algorithm proposed in [1] was successfully implemented in our software engineering project. The algorithm was tested with another state of the art segmentation method on some standard images and both qualitative and qualitative comparisons were made. We incorporated object oriented programming approaches in our implementation and added a graphical user interface. We modified the code so that it can be run using both Qt and CMake and added sample unit test. Multi region segmentation was also successfully implemented.

References

[1] Casaca, W.,Nonato, L. G.,and Taubin, G. (2014, June). *Laplacian Coordinates for Seeded Image Segmentation.* In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on (pp. 384-391). IEEE. Available at http://tinyurl.com/nz77gcmA.

[2] Angelova and S. Zhu. *Efficient object detection and segmentation for fine-grained recognition.* In IEEE Conf. on Comp. Vision and Pattern Recog. (CVPR), pages 811-818, 2013.

[3] B. Wang and Z. Tu. *Affinity learning via self-diffusion for image segmentation and clustering.* In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 23122319, 2012.

[4] L. Grady. *Targeted image segmentation using graph methods.* In O. Lezoray and L. Grady, editors, Image Processing and Analysis with Graphs, pages 111-135. CRC Press, 2012.

[5] C. Couprie, L. Grady, L. Najman, and H. Talbot. *Power watershed: A unifying graph-based optimization framework.* IEEE Trans. Pattern Anal. Mach. Intell., 33(7):13841399, 2011

[6] Y. Boykov and G. Funka-Lea. *Graph cuts and efficient nd image segmentation.* International Journal of Computer Vision, 70(7):109-131, 2006.

[7] L. Grady. *Random walks for image segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(11):1768-1783, 2006.

[8] J. Cousty, G. Bertrand, L. Najman, and M. Couprie. *Watershed cuts: Minimum spanning forests and the drop of water principle.* IEEE Trans. Pattern Anal. Mach. Intell., 31(8):1362-1374, 2009.

[9] O. Sorkine. *Differential representations for mesh processing (the state of the art report).* Computer Graphics Forum (Eurographics), 25(4):789-807, 2006.

[10] K. Xu, H. Zhang, D. Cohen-Or, and Y. Xiong. *Dynamic harmonic fields for surface processing.* Computer and Graphics, 33(3):391398, 2009

[11] V. Jankovic. *Quadratic functions in several variables.* The Teaching of Mathematics, VIII:53-60, 2005

[12] http://opencv.org/

[13] http://eigen.tuxfamily.org/index.php?title=Main_Page

[14] http://eigen.tuxfamily.org/dox/group__TutorialSparse.html

[15] http://eigen.tuxfamily.org/dox/group__TopicSparseSystems.html

[16] http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html

[17] C. Rother, V. Kolmogorov, and A. Blake. *Grabcut: Interactive foreground extraction using iterated graph cuts.* ACM Trans. Graph. (SIGGRAPH 04), 23(3):309-314, 2004.

[18] http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm

[19] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. *Contour detection and hierarchical image segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(5):898-916, 2011.

[20] R. Unnikrishnan, C. Pantofaru, and M. Hebert. *Toward objective evaluation of image segmentation algorithms.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(6):929-944, 2007.

[21] http://www.cns.bu.edu/~lgrady/

[22] http://scikit-image.org/docs/0.10.x/auto_examples/plot_random_walker_segmentation.html

[23] http://scikit-image.org/

[24] https://www.youtube.com/channel/UCYP0nk48grsMwO3iL8YaAKA

[25] https://www.youtube.com/channel/UCJbPGzawDH1njbqV-D5HqKw

[26] http://doc.qt.io/

[27] http://www.qtcentre.org/content/

[28] https://github.com/ViBOT-Erasmus/cmake-qt5-gtest-skeleton