Computer Image Analysis of Malarial *Plasmodium Vivax* in Human Red Blood Cells

A Master’s Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Industrial Engineering

By

André von Mühlen

June 2004
Authorization for Reproduction of Master’s Thesis

Title: Computer Image Analysis of Malarial *Plasmodium Vivax* in Human Red Blood Cells

Author: André von Mühlen

I grant permission for the reproduction of this thesis in its entirety or any of its parts, without further authorization from me.

________________________________
Signature

________________________________
Date
Title: Computer Image Analysis of Malarial *Plasmodium Vivax* in Human Red Blood Cells

Author: André von Mühlen

Date Submitted: 

__________________________   _________________
Dr. Jose Macedo
Committee Chair

__________________________   _________________
Dr. Robert Crockett
Committee Member

__________________________   _________________
Dr. Tali Freed
Committee Member
Abstract

Title: Computer Image Analysis of Malarial *Plasmodium Vivax* in Human Red Blood Cells

Author: André von Mühlen

This work describes a computer image analysis algorithm that counts the number of cells in a digital image, in addition to finding and counting cells with the malarial *P. vivax* parasite. The process is based on morphological operators using flat, disk-shaped, structuring elements to count the number of cells. A morphological operation top-hat filter is used to detect parasites in human red blood cells. The output is 100% sensitive and 83.3% specific to malarial parasites; valuable of a good diagnosis tool for malaria.
Acknowledgements

I would like to thank Dr. Sema Alptekin for encouraging me to pursue a master’s degree in Industrial Engineering, Dr. Jose Macedo for guiding me in writing my thesis, and Dr. Tali Freed for helping me in many of the endeavors that affected my education.

I would also like to thank Carol Peebles at INOVA Diagnostics in San Diego for showing me what a malaria parasite looks like for the first time, Dr. Elena Levine at the Cal Poly Biology Department for assisting me in learning how to use the microscope and digital camera, and the following universities in Brazil that provided the necessary blood smears for analysis: University of Campinas, São Paulo, and PUC in Porto Alegre, Rio Grande do Sul.

Most valuable was the encouragement my parents, Carlos and Denise von Mühlen, gave me to pursue this project, their expertise in the medical field, and assistance in verifying I wrote an accurate description of the malaria disease.
Table of Contents

List of Figures vii
List of Tables viii
1. Introduction 1
   Malaria Life Cycle 1
   Diagnosis 3
   Literature Review 4
   Project Goal 5
2. Methodology 6
   Collection of Materials 6
   Algorithm Design 7
3. Results 19
4. Discussion and Conclusions 21
5. Recommendations for Future Research 23
6. Appendixes
   A — Equipment Specifications 25
   B — Matlab Algorithm 26
   C — Images and Processing 27
   D — Mathematical Algorithms 53
   E — Ratio of Sick and Healthy Cells per Algorithm Output 54
   F — Ratio of Algorithm Cell Count and Actual Cell Count 56
   G — Definition of Algorithm Commands 57
7. References 64
List of Figures

Figure 1. Prevalence of malaria around the world 1
Figure 2. Blood smear infected with *P. vivax* 2
Figure 3. Blood smear with mature schizont 2
Figure 4. Blood smear with gametocytes 3
Figure 5. Thick blood smear at 400X magnification 7
Figure 6. Thin blood smear at 1000X magnification 7
Figure 7. Matlab algorithm overview 8
Figure 8. Dust particles in images of blood smear 9
Figure 9. Dust particles removed from images of blood smear 9
Figure 10. Sample image analyzed by algorithm 9
Figure 11. Inverse of sample image 9
Figure 12. Binary of sample image 10
Figure 13. Presence of holes within cells 11
Figure 14. Image after holes were filled 11
Figure 15. Small elements in sample image 12
Figure 16. Original image (i.e. one cell) 12
Figure 17. Erosion of cell in image 12
Figure 18. Dilation of cell in image 12
Figure 19. Sample image after morphological operation open 13
Figure 20. Sample image after morphological operation open a with radius of 40 pixels 13
Figure 21. Sample image after morphological operation open with a radius of 60 pixels 13
Figure 22. Sample image after clear border command 14
Figure 23. Sample image after top-hat filter 16
Figure 24. Binary of sample image with parasite 16
Figure 25. Sample image with parasite and holes filled 17
Figure 26. Sample image with parasite after morphological operation open 17
List of Tables

Table 1. Results of actual count and algorithm output ........................................ 19
Table 2. Total number of cells and parasites, and pertaining algorithm count total .......... 20
Table 3. Actual presence of parasite vs. algorithm detection .................................... 22
Table 4. Sensitivity, specificity and accuracy of algorithm ...................................... 23
1. Introduction

This project aims to study computer image analysis of malarial parasites using morphological operators as its main method of approach. Malaria is a life-threatening parasitic disease transmitted through female *Anopheles* mosquitoes (1). It is found throughout the tropical and subtropical regions of the world (figure 1), and affects over 300 million people annually. As globalization, ease and frequency of travel increase, malaria cases may occur in any country (2). Over one million people die due to malaria every year (3).

![Figure 1. Prevalence of malaria around the world (Roll Back Malaria. Online. 28 March, 2004)](image)

**Malaria Life Cycle**

Four different species of the parasite of the genus *Plasmodium* cause human malaria: *P. falciparum*, *P. vivax*, *P. malariae*, and *P. ovale* (4). All four have a similar life cycle. An *Anopheles* mosquito feeds on an infected human host and obtains male and female sex cells of the malaria parasite. In the mosquito, these gametocytes mature, and fertilization occurs. An oocyst forms in the stomach of the mosquito and numerous spindle-shaped sporozoites are formed inside. The mature oocyst breaches the stomach wall and spreads throughout the mosquito’s body. When the mosquito feeds, parasites in
the salivary glands are injected into the blood flow of its prey. The sporozoites injected into a human host reach the hepatic parenchymal cells in the liver, where they proliferate. The parasites then leave the liver and infect red cells. In the red cells, the cycle has three stages: trophozoites (growing forms), schizonts (dividing forms), and gametocytes (sexual forms). These stages are described next for the *P. vivax* — the parasite found in the blood films tested for this project.

The youngest trophozoite starts as a ring form with a central vacuole, a red chromatin mass and blue cytoplasm (figure 2, arrow 1). Growing trophozoites beyond the ring stage have more abundant cytoplasm, yet have still one single chromatin mass and may be irregular or compact. Mature trophozoites are usually compact, yet still have only one chromatin mass. Immature schizonts have two or more chromatin masses and an undivided cytoplasm, and mature schizonts have both cytoplasm and chromatin completely divided, so individual merozoites (new parasites) are evident (figure 3). The mature schizont ruptures the cell wall and infects additional red cells.

Gametocytes develop directly from merozoites. Female gametocytes are characterized by a compact chromatin mass, and male gametocytes have chromatin that is more dispersed. Figure 4 has such infected red cells (left and low center with dispersed
red chromatin and blue cytoplasm), healthy red cells (brownish color), and a white cell (large and squiggly in the center). A developing gametocyte is more compact than growing trophozoites; and those of the *P. vivax* have a rounded form.

![Figure 4. Blood smear with gametocytes](image)

1. Female gametocyte
2. Male gametocyte
3. White cell

**Diagnosis**

Malaria is a curable disease and is not an inevitable burden if those suffering from this infection have access to early diagnosis and prompt treatment (3). The “gold standard” for diagnosis involves examination of Giemsa-stained thick and thin blood smears under a microscope by an expert in examination of blood smears for malaria parasites (4-5). This procedure is neither 100% sensitive nor 100% specific, since low-level parasitemia and mixed infections are frequently not detected. One study put the detection limit of microscopy at approximately 50 parasites/µl (6).

Additional methods exist to diagnose malaria, mainly rapid chromatographic assays (similar to a pregnancy test, a color bar shows up on the testing kit if the parasite is present). The main benefits of rapid tests are that they are quick to perform and easy to interpret, and they require no electrical equipment or specialized laboratory facilities (7). However, many negative issues regarding chromatographic tests are of concern: they are expensive (relative to blood films), do not quantify parasite load, have a delayed return to

---

1 A stain that colors DNA only.
negative due to persistent antigenemia following successful treatment, most products cannot detect or differentiate between different plasmodia apart from \textit{P. falciparum}, and they are not sufficiently sensitive at low parasitic densities (8).

**Literature Review**

The development of easy, rapid, and accurate tests for detecting malaria is highly desirable. Using the computer as a diagnostic tool, studies have shown that cancer causing skin lesions are identified reliably by image analysis algorithms, and have the potential to become a useful tool for clinicians (9-11). Counting cells and finding foreign bodies (i.e. bacteria and parasites) in human blood is often challenging, especially if many experiments are involved. Thus, studies have evaluated a variety of methods for algorithms to evaluate sick cells (12-15). To my knowledge, only one paper describing the analysis of images with malarial parasites is found in the literature. The study by Di Ruberto et al. (2002) provides an excellent roadmap into the identification of malaria parasites in red blood, and the three stages the parasites have upon analysis (trophozoites, schizonts, and gametocytes). The authors detect parasites by means of an automatic thresholding based on a morphological approach (15), and propose a morphological method to cell image segmentation based on gray scale granulometries and openings with disk-shaped elements, flat and hemispherical, that they claim is more accurate than the classical watershed-based algorithm (based on segmentation of different elements in an image) (16). Furthermore, they classify the three stages of the parasites by morphological skeleton (i.e. physical differences between each stage of the parasite).

Three other papers were found under image analysis and malaria; however they have different objectives than this project. One study uses malaria images as an example of how a homomorphic filter has valuable properties; thus proving the author’s claim of
simultaneous dynamic range compression and balanced contrast enhancement of computer images (17). The study is strictly about image filters, not malaria.

The second study analyzes computer images of Chondroitin Sulfate A (CSA) expression in placentas with *P. falciparum*. The authors found that the presence of malaria parasite was significantly correlated with immunostaining of CSA (12). This study did not analyze images of the parasites alone.

The third paper analyzes the calcium content of cells infected with malaria parasites (18). The study used digital image analysis of a fluorescent indicator that has different visible properties according to the amount of calcium concentrations inside the cells. Based on the indicator fluorescence and the aid of a calibration of the calcium response, the study concluded that free calcium concentrations in the intact parasite are maintained at a predetermined level, regardless of the free calcium in the surrounding area. Again, this study did not use the technology to look for parasites.

**Project Goal**

This project sets out to develop and test a software algorithm that counts the number of healthy and sick cells in a set of images. Unlike microscopy and rapid chromatographic assays, the software should be able to find parasites in any number or quantity of blood; and the results should come out within minutes. Nonetheless, there are factors that influence the algorithm design that are not critical issues in the diagnosis methods described earlier: lighting, cell size in images, spacing between cells, and physical properties of parasites and white cells. The method of approach to counting cells is similar to Di Ruberto and collaborators (2002), using a structural disk-shaped element and morphological operations to outline red cells; however, the method to finding the parasites is unique to a filter that delineates the morphology of the physical elements that make up the parasite inside a red cell.
2. Methodology

This project was divided in seven parts: literature review, collection of materials, algorithm design, collection of data, discussion of results, conclusions and suggestions for future research.

Collection of Materials

A set of Giemsa-stained blood films was obtained from Brazilian universities for analysis. Six slides had malaria-infected blood smears, and eleven slides had healthy blood smears, which were used as a reference.

Digital photographs were taken at the Cal Poly Biology Department, under the supervision of Dr. Elena Levine, with an Olympus microscope BX60 and digital camera DEI-760D (see appendix A for specifications). Magnification was set at 1,000X and light settings varied between slides and individual photographs due to the number of cells in each shot, and the amount of lighting coming through the blood plasma from the microscope.

Most of the images were taken in the Portable Network Graphics (PNG) format (very high resolution and no loss in quality when saving, restoring and re-saving an image) (19), while the remaining were taken in the Joint Photographic Experts Group (JPEG) format (same resolution yet compressed more). The JPEGs were formatted to PNG before analysis to maintain consistency in how Matlab 6.5 and its Image Processing Toolbox module handles images, fills holes, converts to binary, etc. All images had 1280x1024 pixels.

All infected blood films except two were thick blood smears. These are good for microscopy diagnosis since many cells are visible. However, cells are also clustered together; which makes it very difficult for the computer to differentiate. Figures 5 and 6 illustrate the difference between thick and thin blood smears.
Of the remaining two, one was in very poor shape (few cells present) and was discarded. The other had a large number of cells present, with good spacing between each. The last blood film was used to obtain the 32 images used for analysis — 16 with parasites and 16 without parasites in the image.

The set of healthy blood slides was used exclusively for comparison of sick and healthy cells. No images were taken.

**Algorithm Design**

The algorithm for this project has to consider a number of factors that affect both the overall cell count, and the parasite count. All cells in the images have a physical border around them; however they may not always be clear in digital images, since a large amount of light passes through their bodies. Furthermore, the cells in the pictures are not flat on the slide, but floating in plasma; hence some cells are actually on their side. Due to their 3-dimensional state, some portions of cells will not look as round as one might expect to see. The lack of uniform physical shape means the size of cells varies from 30-100 pixels.
The amount of magnification also has an effect on the algorithm. Since some cells are deeper in blood plasma than others, and the depth of focus of the microscope is very small at 1000X (0.69 µm), some cells are out of focus on the images.

Figure 7 provides an overview of the algorithm created to process the number of cells and parasites.

Figure 7. Matlab algorithm overview

**Inverse of Image**

The original digital images had a few dust particles from the lenses in the microscope. Since these were present in the exact same place in every image, and could cause errors in the cell count, they were removed prior to processing. Photoshop’s Clone Stamp tool was used for this task, which stamps nearby pixels onto the space in question (figures 8 and 9). This process was done manually.
After the dust particles were removed, the inverse of all images was taken (i.e. black turns to white, red turns to blue, etc.). This step is required so that points of interest (i.e. cells) are offset from the background prior to converting the image to binary (figures 10 and 11).

After reading the images into Matlab, the algorithm can process the cell and parasite count. For additional information on each command used, see appendix G; which contains a verbatim description of each command by the software company, MathWorks.

**Level Threshold**

This step precedes turning the image to binary, as it provides the global threshold level information needed for the command to highlight bright pixels as white, and dark pixels as black (20). Since the cells are lighter than the background, they receive a value
of 1. The dark background receives a value of 0. There is no output for this command. Matlab keeps the information in the computer’s memory, under the name level.

The line command is `level = graythresh(I)`. The term I stands for the inverse image.

One limitation of this command is that a cell exceptionally out of focus may not be bright enough, and therefore not discernible from the background. When the image is turned to binary, the cell is dropped.

**Color to Binary**

The color image is then processed to black and white (figure 12). The line command is `bw = im2bw(I,level)`, wherein `im2bw` is the Matlab command that calls image I with intensity information level discussed above. The term bw stands for a black and white image.

![Figure 12. Binary of sample image](image)

**Fill Holes**

Some cells will show up with holes within them (figure 13, red arrows). The cause may be dark spots where not enough light came through, borders that are out of focus, or irregular elements inside, such as a malaria parasite or DNA in a white cell. To prevent these holes from causing cells to be dropped due to sizes smaller than usual (a red
cell has approximately 40-60 pixels in these images; white cells are larger), a command is used to fill them.

The line command is \texttt{bw2 = imfill(bw,'holes')}, wherein \texttt{imfill} is the Matlab command, \texttt{bw} is the binary image, \texttt{'holes'} tells Matlab to look for openings, cracks or fissures within bodies (versus filling the entire image), and \texttt{bw2} is the output (figure 14).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/holes.png}
\caption{Figure 13. Presence of holes within cells}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/holes_filled.png}
\caption{Figure 14. Image after holes were filled}
\end{figure}

Open

The image at this point still has a few elements that are not relevant, including parts of cells on the periphery of the image (figure 15, arrow 1), and other small elements unknown to the author (arrows 2 and 3). To eliminate these items, a morphological opening is performed, consisting of an erosion and a dilation of the elements on the image (21-22). An essential part of these two operations is the structuring element (named \texttt{strel} in the code) used to probe the input image (23). A two-dimensional, or flat, structuring element consists of a matrix of 0's and 1's. The center pixel of the structuring element, called the origin, identifies the pixel of interest — the pixel being processed. The pixels in the structuring element containing 1's define the neighborhood of the structuring element. These pixels are also considered in erosion or dilation processing. A disk function is used to outline the shape of the cells — what the software is supposed to examine, and a radius (number of pixels) is input as the minimum size of the neighborhood. Anything smaller is eliminated.
After the neighborhood of pixels is defined, an erosion operation is performed that modifies all pixels set as 1’s with the value 0, per the radius set by the user. See the illustration in figures 16 and 17.

The next step of the open is a dilation of the image. See illustration in figure 18.

The command line is `background = imopen(bw2,strel('disk',20));`
The term background will carry the new image to the next step. See figure 19, the new image, clear of any elements smaller than 20 pixels. Notice how the cells have more of a rounded form. Small inlays on the edges are either erased or enlarged.

Figure 19. Sample image after morphological operation open

A radius larger than 20 pixels will drop more elements (i.e. those two cells on the bottom right of the image) and will also assist in separating connected cells (see top right figures 19 and 20); however, it also causes an increased effect on the morphology of the cells. Smaller, non-circular cells also drop out. See figures 20 and 21, for a pixel radius of 40 and 60, respectively.

Figure 20. Sample image after morphological operation open a with radius of 40 pixels

Figure 21. Sample image after morphological operation open with a radius of 60 pixels

The radius value of 20 pixels provides the optimum count when one wants to use the exact same algorithm code for the entire set of images. Note that this value only
applies for the images analyzed for this project. Images taken with a different blood film preparation, magnification, microscope, camera, set of lenses, etc. may have cells of different sizes which require a different radius value.

**Drop Cells on Borders**

Since it is often difficult to differentiate which type of cell is present on the periphery of an image, the algorithm removes them.

The command line is `nobrd = imclearborder(background,26)`. The largest distance from the border that Matlab allows for analysis is 26 pixels; hence some cells stay in the output (figure 22). Another limitation is observed in figures 19 and 22: cells away from the border which are interconnected to cells on the border are also dropped.

![Figure 22. Sample image after clear border command](image)

A third limitation involves the equipment used to take the images. One can notice a thin black slit between the left side of the image above and the cells next to it. Since the cells are not actually connected to the border (due to a lapse by the Olympus hardware/software), they are never analyzed by the clear border command.

**Count Cells**

The Euler number command is used to count the number of cells. It functions by considering patterns of convexity and concavity in local 2-by-2 pixel neighborhoods (20).
The command line is `eul = bweuler(nobrd)`. The output for the image above is the `eul` value 28. A manual count of the image indicates there are in fact 28 cells present.

A pre-count of each image was performed manually where only the cells not touching the periphery of the image were considered. The sum for the image above (number four in appendix C) was 28. Hence, while the algorithm erred in keeping certain cells on the borders and dropping other interconnected cells, it still returned the true count for this image. The implications of this output are detailed in the results and discussion portion of the project.

**Morphological Structuring**

This step involves creating a structuring element with a 25-pixel disk-shaped radius of the inverse image. The command line is `structure = strel('disk',25);`

The information `structure` will be used on the next step.

Two types of cells were found in the images: red and white cells. Red cells carry oxygen from the lungs to the tissues around the body, and carbon dioxide from the tissues to the lungs. White cells are responsible for protecting the body from germs or infections (24-25). They do so by releasing protective antibodies that overpowers the germ, or surround and devour the bacteria. The computer algorithm has to differentiate between these two types of cells, since both infected red cells and white cells have DNA; which, when stained, have the same color. Healthy red cells do not have DNA (figure 4). The algorithm takes into account the different physical shapes of each cell by performing a morphological top-hat filter on the inverse image using the structuring element `structure` above.

**Top-Hat Filtering**

As the open removes narrow protrusions or spikes on the contour of cells, and tiny elements altogether, the top-hat filter reveals exactly these residues that the
structuring element does not fit (26); hence, all the small elements inside sick red cells come up stronger in the output than healthy red cells (figure 23), or the edges of white cells (see appendix C, images 13 and 14).

Figure 23. Sample image after top-hat filter

The line command is $I_{top} = \text{imtophat}(I, \text{structure})$;

**Level Threshold**

At this point, the algorithm needs the global threshold value discussed earlier to convert to binary. The line command is $\text{level} = \text{graythresh}(I_{top})$;

**Color to Binary**

Using the same method discussed earlier, the image is converted to binary (figure 24).

Figure 24. Binary of sample image with parasite
The line command is \( \text{bw} = \text{im2bw}([\text{top}],[\text{level}]); \)

**Fill Holes**

The holes are then filled to prevent the parasite from being dropped along with other smaller elements during the *open* (figure 25).

![Figure 25. Sample image with parasite and holes filled](image)

The line command is \( \text{bw2} = \text{imfill}([\text{bw}],'\text{holes}'); \)

This command also has limitations. Depending on the thickness of the edge of a red cell, the fill hole command fills a healthy cell’s outline. Image 10 in appendix C shows how a healthy cell out of focus can be counted as sick.

**Open**

Figure 26 illustrates the result of an open with a pixel radius of 26.

![Figure 26. Sample image with parasite after morphological operation open](image)
A smaller radius will not drop enough of the remnant elements on the image; while a larger radius will drop the smaller sick cells found in some images (see image 12 in appendix C).

The command line is `background2 = imopen(bw2, strel('disk', 26));`

**Count Parasites**

Last step in the algorithm is an Euler number command. The command line is `eul = bweuler(background2).` The output is 1; which is correct.

See appendix B for the entire algorithm code, and an index for each image in appendix C. Appendix D has mathematical algorithms for the morphological operations open and top-hat filter.
3. Results

Table 1 shows the number of cells in each image per a manual count by the author (where only cells within the image boundaries are considered), whether there is a parasite or not in each image, the algorithm output for the total number of cells (includes white cells, and healthy and sick red cells), the actual number of sick cells, and the algorithm output for the number of sick cells.

<table>
<thead>
<tr>
<th>Photo Number</th>
<th>Sick (YES/NO)</th>
<th>Actual Count: Total Number of Cells</th>
<th>Algorithm Count: Total Number of Cells</th>
<th>Actual Number of Sick Cells</th>
<th>Algorithm Count of Sick Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO</td>
<td>25</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>YES</td>
<td>25</td>
<td>27</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>YES</td>
<td>24</td>
<td>24</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>YES</td>
<td>28</td>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>YES</td>
<td>24</td>
<td>30</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>NO</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>NO</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>NO</td>
<td>25</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>YES</td>
<td>13</td>
<td>14</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>YES</td>
<td>23</td>
<td>25</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>YES</td>
<td>30</td>
<td>30</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>YES</td>
<td>25</td>
<td>22</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>NO</td>
<td>21</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>NO</td>
<td>20</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>YES</td>
<td>34</td>
<td>37</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>YES</td>
<td>28</td>
<td>29</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>NO</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>NO</td>
<td>21</td>
<td>21</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>NO</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>NO</td>
<td>15</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>YES</td>
<td>23</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>NO</td>
<td>22</td>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>YES</td>
<td>18</td>
<td>18</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>YES</td>
<td>18</td>
<td>14</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>YES</td>
<td>27</td>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>YES</td>
<td>25</td>
<td>25</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1. Results of actual count and algorithm output
From the initial 32 images taken, only 26 were analyzed. Six images were dropped because the algorithm could not process them, leading to inconclusive results. There are three groups of reason that influenced this decision: blood film preparation, quality of images, and algorithm design.

Blood film preparation considers lack of spacing between cells, and other elements in the images the author could not identify. The better the spacing, the more accurate the count.

Quality of images considers focus, and too much or too little lighting through cells; hence they blend with the background. A set of images where every cell is in focus provides for better results.

Algorithm design considers limitations by the algorithm that should be worked out in future research.

Table 2 shows the total number of cells in all the images, the algorithm count, the total number of sick cells, and the pertaining algorithm count. It also lists the total number of cells missed by the algorithm, healthy and sick. This information is used in the following equations to count the error rates of the algorithm for percentage of cells missed and over counted, and percentage of sick cells missed and over counted.

<table>
<thead>
<tr>
<th>Actual Count: Total Number of Cells</th>
<th>Algorithm Count: Total Number of Cells</th>
<th>Total Number of Cells Missed</th>
<th>Actual Number of Sick Cells</th>
<th>Algorithm Count of Sick Cells</th>
<th>Total Number of Sick Cells Missed</th>
</tr>
</thead>
<tbody>
<tr>
<td>584</td>
<td>589</td>
<td>19</td>
<td>17</td>
<td>22</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2. Total number of cells and parasites, and pertaining algorithm count total

**Percentage of cells missed** = \( \frac{\text{total number of cells missed}}{\text{total number of cells}} \) = \( \frac{19}{584} \) = 3.3%
The algorithm missed 3.3% of the cells due to physical contact between cells. A lack of discernible separation between two cells is counted as one cell by the software. The border drop command also dropped whole cells touching partial cells on the border.

\[
\text{Percentage of cells over counted} = \frac{\text{number of cells over counted}}{\text{total number of cells}} = \frac{77}{584} = 13.2\%
\]

Cells over counted came from the border of the images and small unidentified elements. These were not counted in the manual count, hence the disparity of 13.2%.

\[
\text{Percentage of sick cell cells missed} = \frac{\text{total number of sick cells missed}}{\text{total number of cells}} = \frac{1}{17} = 5.9\%
\]

The algorithm missed one parasite in image number 12. The cell had very dispersed elements and its binary cell image did not fill with the fill-hole command, hence it was dropped after the open. Note that another healthy cell was recognized as sick in the same image; thus, the correct count output for this image in table 1.

\[
\text{Percentage of sick cells over counted} = \frac{\text{number of sick cells over counted}}{\text{total number of sick cells}} = \frac{5}{17} = 29.4\%
\]

The algorithm erred in the count of sick cells by almost 30% over the actual number. The use of blood smears prepared for computer image analysis would decrease this number, due to overall better image quality. The blood smears available for this study were prepared solely for visual microscopy by an expert in malarial analysis. More emphasis on spacing between cells and an overall flat state by cells against the glass on the background would diminish errors.

Appendix E has a table that quantifies the extent of the disease in each image with a ratio of sick cells and the total number of cells; and appendix F has a table that provides a quality analysis of the cell count with a ratio of the algorithm count and the actual cell count by the author.
4. Discussion and Conclusions

This study found that the use of computer image analysis of digital images to count the number of cells and detect parasites in blood smears provides a valid diagnosis tool for malaria. Di Ruberto and collaborators were the first researchers to use computer image vision to detect malaria parasites. They used Matlab to count the number of cells by use of gray scale granulometries, and detected parasites by means of an automatic thresholding based on a morphological approach. In similar manner, this project uses morphological operators to count the number of cells; however, it uses a top-hat filter to detect parasites within cells. Both methods seem to provide similar results.

Table 4 outlines the sensitivity, specificity and accuracy of the algorithm — defined by the equations below (27). In table 3, the term *positive microscopy* represents the number of images that contain the malaria parasite; while negative microscopy represents images with only healthy cells. Algorithm positive represents images that the algorithm found a parasite, whether it was right or not; and algorithm negative represents images the algorithm did not find any parasite.

<table>
<thead>
<tr>
<th>Algorithm Microscopy</th>
<th>Positive Microscopy</th>
<th>Negative Microscopy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive Algorithm</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>Negative Algorithm</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Table 3. Actual presence of parasite vs. algorithm detection

**Sensitivity** = \( \frac{a}{a + c} \)

**Specificity** = \( \frac{d}{b + d} \)

**Accuracy** = \( \frac{(a + d)}{(a + b + c + d)} \)
<table>
<thead>
<tr>
<th>Parasite</th>
<th>Positive Microscopy</th>
<th>Negative Microscopy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive Algorithm</td>
<td>Positive Algorithm</td>
</tr>
<tr>
<td></td>
<td>Negative Algorithm</td>
<td>Negative Algorithm</td>
</tr>
<tr>
<td></td>
<td>Sensitivity (%)</td>
<td>Specificity (%)</td>
</tr>
<tr>
<td></td>
<td>Accuracy (%)</td>
<td></td>
</tr>
<tr>
<td>$P.\ vivax$</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>100.0</td>
<td>83.3</td>
</tr>
<tr>
<td></td>
<td>92.3</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Sensitivity, specificity and accuracy of algorithm

The algorithm has high sensitivity, which is desirable since one would like a test of high success rate of detecting positive cases. Inability of the test to correctly identify a true positive might result in missing potential cases of the disease. Conversely, the test is not highly specific at 83.3%, hence there are false positives. Overall, the algorithm had an accuracy of 92.3%, which is considered good under clinical parameters.

The algorithm developed to find and count parasites is a novel alternative to current diagnostic methods. The software is able to find parasites in low quantities and quantify the extent of infection, all in less than five minutes. However, the use of computer image vision requires expensive equipment not available in many countries, mainly those affected by the epidemics. Microscopy continues to be the gold standard for the diagnosis of malaria.
5. Recommendations for Future Research

As outlined in table 1, the algorithm over counted and under counted the number of cells in certain images. It also counted some healthy red and white cells as sick with parasites. To increase the accuracy of the output, and diminish type I and II errors, the author suggests an algorithm that collects an array of results based on a large number of cycles. The algorithm would adjust one factor per cycle, and compare the output with all the previous runs until the results maintained some consistency; hence it would stop only when it reached a pre-determined consensus value (i.e. ten equal cell counts). Factors would include image brightness and contrast, open radius, structuring element radius for the top-hat filter, and perhaps a color overlay for the inverse image. Shades of blue, as seen in figure 11, might not provide the best basis for collection of thresholding information to binary.

Furthermore, the preparation of the blood smears needs to take into account the purpose of the sample. The better the spacing between cells, the less likely the chance of type I and II errors; where multiple cells were counted as one, and one or more cells were dropped. Images out of focus should be discarded; and either the imaging equipment should not leave gaps on the border of images, or the gap between cells and the border should be automatically shortened.
Appendix A — Equipment Specifications

System Microscope

Olympus BX60
Lenses: Olympus 10X, 20X, 40X, 100X

Optical Characteristics:

<table>
<thead>
<tr>
<th>Magnification</th>
<th>Resolution (µm)</th>
<th>Total Magnification</th>
<th>Depth of Focus (µm)</th>
<th>Field of View</th>
</tr>
</thead>
<tbody>
<tr>
<td>10X</td>
<td>1.34</td>
<td>100X</td>
<td>28.0</td>
<td>2.2</td>
</tr>
<tr>
<td>20X</td>
<td>0.84</td>
<td>200X</td>
<td>6.09</td>
<td>1.1</td>
</tr>
<tr>
<td>40X</td>
<td>0.52</td>
<td>400X</td>
<td>3.04</td>
<td>0.55</td>
</tr>
<tr>
<td>100X</td>
<td>0.27</td>
<td>1000X</td>
<td>0.69</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Digital Camera

Olympus DEI-750D Digital Output
Model 60660
Image Sensor: 3 x ½” CCD-chip, one per color channel
Picture Format: 6.4 mm x 4.8 mm
Internal lens: f 1.4, primary color (RGB)
Picture Elements (NTSC): 768 x 582 pixels per chip, 1,138,176 in total

Software

Adobe Photoshop 6.5 (for dust removal and image inverse)
Microsoft Excel 2003
The MathWorks Matlab 6.5.0.1924 Release 13
Image Processing Toolbox 4.1
**Appendix B — Matlab Algorithm**

<table>
<thead>
<tr>
<th>Code</th>
<th>Image Output (see appendix C)</th>
<th>Count Output (see appendix C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I = imread('filename.png');</td>
<td></td>
<td></td>
</tr>
<tr>
<td>level = graythresh(I);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw = im2bw(I,level);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(bw)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>bw2 = imfill(bw,'holes')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(bw2)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>background = imopen(bw2,strel('disk',20));</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(background)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nobrd = imclearborder(background,26)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(nobrd)</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>eul = bweuler(nobrd)</td>
<td>4</td>
<td>Cell Count</td>
</tr>
<tr>
<td>structure = strel('disk', 25);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Itop = imtophat(I, structure);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(Itop, [])</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>level = graythresh(Itop);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw = im2bw(Itop,level);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(bw)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw2 = imfill(bw,'holes')</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>figure, imshow(bw2)</td>
<td>7</td>
<td>Parasites Count</td>
</tr>
<tr>
<td>background2 = imopen(bw2,strel('disk',26));</td>
<td></td>
<td></td>
</tr>
<tr>
<td>figure, imshow(background2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eul = bweuler(background2)</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C — Images and Processing

Image 1:

Cell Count: 24
Actual: 25

Parasites: 0
Actual: 0
Image 2:

Count: 27
Actual: 25

Parasites: 1
Actual: 1
Image 3:

Count: 24
Actual: 24

Parasites: 1
Actual: 1
Image 5:

Count: 30
Actual: 24

Parasites: 1
Actual: 1
Image 7:

Count: 15
Actual: 15

Parasites: 0
Actual: 0
Image 8:

Count: 24
Actual: 25

Parasites: 0
Actual: 0
Image 12:

Count: 22
Actual: 25

Parasite: 2
Actual: 2
Image 13:

Count: 17
Actual: 21

Parasites: 0
Actual: 0
Image 18:

Count: 21
Actual: 21

Parasites: 1
Actual: 0
Image 21:

Count: 28
Actual: 23

Parasites: 0
Actual: 0
Image 24:

Count: 14
Actual: 18

Parasites: 2
Actual: 2
Appendix D — Mathematical Algorithms

Open

The eroded image of an object $O$ with respect to a structuring element $S$ with a reference point $R$, \( O \ominus S \), is the set of all reference points for which $S$ is completely contained in $O$.

The dilated image of an object $O$ with respect to a structuring element $S$ with a reference point $R$, \( O \oplus S \), is the set of all reference points for which $O$ and $S$ have at least one common point.

Opening is defined as an erosion, followed by a dilation: \( (O \ominus S) \oplus S \).

Closing is defined as a dilation, followed by an erosion: \( (O \oplus S) \ominus S \).

Reference


Top-Hat Filter

The isolation of gray-value objects that are convex can be accomplished with the top-hat transform. Depending upon whether we are dealing with light objects on a dark background or dark objects on a light background, the transform is defined as:

Light objects - \( \text{TopHat}(\hat{A}, B) = \hat{A} - (\hat{A} \ominus E) = \hat{A} - \max_b \left( \min_b \hat{A} \right) \)

Dark objects - \( \text{TopHat}(\hat{A}, B) = (\hat{A} \cdot B) - \hat{A} = \min_b \left( \max_b \hat{A} \right) - \hat{A} \)

where the structuring element $B$ is chosen to be bigger than the objects in question and, if possible, to have a convex shape. Because of the properties given in equations and, \( \text{TopHat}(A,B) > 0 \). An example of this technique is shown in the following figure.

The original image including shading is processed by a 15 x 1 structuring element as described in eqs. and to produce the desired result. Note that the transform for dark objects has been defined in such a way as to yield "positive" objects as opposed to "negative" objects. Other definitions are, of course, possible.

Thresholding

A simple estimate of a locally-varying threshold surface can be derived from morphological processing as follows:

Threshold surface - \( \bar{\theta}[m,n] = \frac{1}{2} \left( \max(A) + \min(A) \right) \)
Once again, we suppress the notation for the structuring element $B$ under the max and min operations to keep the notation simple. Its use, however, is understood.

Reference

<http://www.ph.tn.tudelft.nl/Courses/FIP/frames/fip-Segmenta.html#Heading119>
Appendix E — Ratio of Algorithm Cell Count and Actual Sick Cell Count

<table>
<thead>
<tr>
<th>Photo Number</th>
<th>Sick (YES/NO)</th>
<th>Algorithm Count of Total Number of Cells</th>
<th>Algorithm Count of Sick Cells</th>
<th>Extent of Disease Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO</td>
<td>24</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>YES</td>
<td>27</td>
<td>1</td>
<td>3.70%</td>
</tr>
<tr>
<td>3</td>
<td>YES</td>
<td>24</td>
<td>1</td>
<td>4.17%</td>
</tr>
<tr>
<td>4</td>
<td>YES</td>
<td>28</td>
<td>1</td>
<td>3.57%</td>
</tr>
<tr>
<td>5</td>
<td>YES</td>
<td>30</td>
<td>1</td>
<td>3.33%</td>
</tr>
<tr>
<td>6</td>
<td>NO</td>
<td>20</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>7</td>
<td>NO</td>
<td>15</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>8</td>
<td>NO</td>
<td>24</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>9</td>
<td>YES</td>
<td>14</td>
<td>1</td>
<td>7.14%</td>
</tr>
<tr>
<td>10</td>
<td>YES</td>
<td>25</td>
<td>2</td>
<td>8.00%</td>
</tr>
<tr>
<td>11</td>
<td>YES</td>
<td>30</td>
<td>1</td>
<td>3.33%</td>
</tr>
<tr>
<td>12</td>
<td>YES</td>
<td>22</td>
<td>2</td>
<td>9.09%</td>
</tr>
<tr>
<td>13</td>
<td>NO</td>
<td>17</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>14</td>
<td>NO</td>
<td>17</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>15</td>
<td>YES</td>
<td>37</td>
<td>2</td>
<td>5.41%</td>
</tr>
<tr>
<td>16</td>
<td>YES</td>
<td>29</td>
<td>1</td>
<td>3.45%</td>
</tr>
<tr>
<td>17</td>
<td>NO</td>
<td>16</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>18</td>
<td>NO</td>
<td>21</td>
<td>1</td>
<td>4.76%</td>
</tr>
<tr>
<td>19</td>
<td>NO</td>
<td>19</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>20</td>
<td>NO</td>
<td>17</td>
<td>2</td>
<td>11.76%</td>
</tr>
<tr>
<td>21</td>
<td>YES</td>
<td>28</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>22</td>
<td>NO</td>
<td>20</td>
<td>0</td>
<td>NA</td>
</tr>
<tr>
<td>23</td>
<td>YES</td>
<td>18</td>
<td>1</td>
<td>5.56%</td>
</tr>
<tr>
<td>24</td>
<td>YES</td>
<td>14</td>
<td>2</td>
<td>14.29%</td>
</tr>
<tr>
<td>25</td>
<td>YES</td>
<td>28</td>
<td>1</td>
<td>3.57%</td>
</tr>
<tr>
<td>26</td>
<td>YES</td>
<td>25</td>
<td>2</td>
<td>8.00%</td>
</tr>
</tbody>
</table>

The ratio is wrong for images 18 and 20; since there are, in fact, no parasites in those images.

**Extent of Disease Ratio** =  \( \frac{\text{Algorithm Count of Sick Cells}}{\text{Algorithm Count}} \)
Appendix F — Ratio of Algorithm Count and Actual Cell Count

<table>
<thead>
<tr>
<th>Photo Number</th>
<th>Sick (YES/NO)</th>
<th>Actual Count</th>
<th>Algorithm Count</th>
<th>Cell Count Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NO</td>
<td>25</td>
<td>24</td>
<td>96.00%</td>
</tr>
<tr>
<td>2</td>
<td>YES</td>
<td>25</td>
<td>27</td>
<td>108.00%</td>
</tr>
<tr>
<td>3</td>
<td>YES</td>
<td>24</td>
<td>24</td>
<td>100.00%</td>
</tr>
<tr>
<td>4</td>
<td>YES</td>
<td>28</td>
<td>28</td>
<td>100.00%</td>
</tr>
<tr>
<td>5</td>
<td>YES</td>
<td>24</td>
<td>30</td>
<td>125.00%</td>
</tr>
<tr>
<td>6</td>
<td>NO</td>
<td>20</td>
<td>20</td>
<td>100.00%</td>
</tr>
<tr>
<td>7</td>
<td>NO</td>
<td>15</td>
<td>15</td>
<td>100.00%</td>
</tr>
<tr>
<td>8</td>
<td>NO</td>
<td>25</td>
<td>24</td>
<td>96.00%</td>
</tr>
<tr>
<td>9</td>
<td>YES</td>
<td>13</td>
<td>14</td>
<td>107.69%</td>
</tr>
<tr>
<td>10</td>
<td>YES</td>
<td>23</td>
<td>25</td>
<td>108.70%</td>
</tr>
<tr>
<td>11</td>
<td>YES</td>
<td>30</td>
<td>30</td>
<td>100.00%</td>
</tr>
<tr>
<td>12</td>
<td>YES</td>
<td>25</td>
<td>22</td>
<td>88.00%</td>
</tr>
<tr>
<td>13</td>
<td>NO</td>
<td>21</td>
<td>17</td>
<td>80.95%</td>
</tr>
<tr>
<td>14</td>
<td>NO</td>
<td>20</td>
<td>17</td>
<td>85.00%</td>
</tr>
<tr>
<td>15</td>
<td>YES</td>
<td>34</td>
<td>37</td>
<td>108.82%</td>
</tr>
<tr>
<td>16</td>
<td>YES</td>
<td>28</td>
<td>29</td>
<td>103.57%</td>
</tr>
<tr>
<td>17</td>
<td>NO</td>
<td>16</td>
<td>16</td>
<td>100.00%</td>
</tr>
<tr>
<td>18</td>
<td>NO</td>
<td>21</td>
<td>21</td>
<td>100.00%</td>
</tr>
<tr>
<td>19</td>
<td>NO</td>
<td>19</td>
<td>19</td>
<td>100.00%</td>
</tr>
<tr>
<td>20</td>
<td>NO</td>
<td>15</td>
<td>17</td>
<td>113.33%</td>
</tr>
<tr>
<td>21</td>
<td>YES</td>
<td>23</td>
<td>28</td>
<td>121.74%</td>
</tr>
<tr>
<td>22</td>
<td>NO</td>
<td>22</td>
<td>20</td>
<td>90.91%</td>
</tr>
<tr>
<td>23</td>
<td>YES</td>
<td>18</td>
<td>18</td>
<td>100.00%</td>
</tr>
<tr>
<td>24</td>
<td>YES</td>
<td>18</td>
<td>14</td>
<td>77.78%</td>
</tr>
<tr>
<td>25</td>
<td>YES</td>
<td>27</td>
<td>28</td>
<td>103.70%</td>
</tr>
<tr>
<td>26</td>
<td>YES</td>
<td>25</td>
<td>25</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Cell Count Ratio = \( \frac{\text{Algorithm Count}}{\text{Actual Count}} \)
Appendix G — Definition of Algorithm Commands

The following command descriptions come from MathWorks’ web site: <http://www.mathworks.com>

graythresh

Compute global image threshold using Otsu’s method

**Syntax**

- `level = graythresh(I)`

**Description**

`level = graythresh(I)` computes a global threshold (`level`) that can be used to convert an intensity image to a binary image with `im2bw`.

The `graythresh` function uses Otsu's method, which chooses the threshold to minimize the intraclass variance of the black and white pixels.

**Class Support**

The input image `I` can be of class `uint8`, `uint16`, or `double` and it must be nonsparse. The return value `level` is a double scalar.

**Reference**


im2bw

Convert an image to a binary image, based on threshold

**Syntax**

- `BW = im2bw(I,level)`

**Description**

`im2bw` produces binary images from indexed, intensity, or RGB images. To do this, it converts the input image to grayscale format (if it is not already an intensity image), and then uses thresholding to convert this grayscale image to binary. The output binary image `BW` has values of 0 (black) for all pixels in the input image with luminance less than `level` and 1 (white) for all other pixels. (Note that you specify `level` in the range [0,1], regardless of the class of the input image.)

`BW = im2bw(I,level)` converts the intensity image `I` to black and white.

**Note** The function `graythresh` can be used to compute the level argument automatically.
Class Support

The input image can be of class `uint8`, `uint16`, or `double` and it must be nonsparse. The output image, `BW`, is of class `logical`.

**imfill**

Fill image regions

**Syntax**

- `BW2 = imfill(BW,locations)`
- `BW2 = imfill(BW,'holes')`
- `I2 = imfill(I)`

**Description**

`BW2 = imfill(BW,locations)` performs a flood-fill operation on background pixels of the binary image `BW`, starting from the points specified in `locations`. If `locations` is a P-by-1 vector, it contains the linear indices of the starting locations. If `locations` is a P-by-ndims(`BW`) matrix, each row contains the array indices of one of the starting locations.

`BW2 = imfill(BW,'holes')` fills holes in the binary image `BW`. A hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image.

`I2 = imfill(I)` fills holes in the intensity image `I`. In this case, a hole is an area of dark pixels surrounded by lighter pixels.

**Class Support**

The input image can be numeric or logical, and it must be real and nonsparse. It can have any dimension. The output image has the same class as the input image.

**Algorithm**

`imfill` uses an algorithm based on morphological reconstruction.

**Reference**


**imopen**

Open an image

**Syntax**

- `IM2 = imopen(IM,SE)`
- `IM2 = imopen(IM,NHOOD)`
**Description**

\[ IM2 = \text{imopen}(IM, SE) \]
performs morphological opening on the grayscale or binary image \( IM \) with the structuring element \( SE \). The argument \( SE \) must be a single structuring element object, as opposed to an array of objects.

\[ IM2 = \text{imopen}(IM, \text{NHOOD}) \]
performs opening with the structuring element \( \text{strel}(\text{NHOOD}) \), where \( \text{NHOOD} \) is an array of 0’s and 1’s that specifies the structuring element neighborhood.

**Class Support**

\( IM \) can be any numeric or logical class and any dimension, and must be nonsparse. If \( IM \) is logical, then \( SE \) must be flat. \( IM2 \) has the same class as \( IM \).

**imclearborder**

Suppress light structures connected to image border

**Syntax**

- \( IM2 = \text{imclearborder}(IM) \)

**Description**

\[ IM2 = \text{imclearborder}(IM) \]
suppresses structures that are lighter than their surroundings and that are connected to the image border. \( IM \) can be an intensity or binary image. The output image, \( IM2 \), is intensity or binary, respectively. The default connectivity is 8 for two dimensions, 26 for three dimensions, and \( \text{conndef}(\text{ndims}(BW), \text{'maximal'}) \) for higher dimensions.

**Note**
For intensity images, \text{imclearborder} tends to reduce the overall intensity level in addition to suppressing border structures.

**Class Support**

\( IM \) can be a numeric or logical array of any dimension, and it must be nonsparse and real. \( IM2 \) has the same class as \( IM \).

**Algorithm**

\text{imclearborder} uses morphological reconstruction where

- Mask image is the input image.
- Marker image is zero everywhere except along the border, where it equals the mask image.

**Reference**

bweuler

Compute the Euler number of a binary image

Syntax

- eul = bweuler(BW,n)

Description

`eul = bweuler(BW,n)` returns the Euler number for the binary image `BW`. The return value `eul` is a scalar whose value is the total number of objects in the image minus the total number of holes in those objects. The argument `n` can have a value of either 4 or 8, where 4 specifies 4-connected objects and 8 specifies 8-connected objects; if the argument is omitted, it defaults to 8.

Class Support

`BW` can be numeric or logical and it must be real, nonsparse, and two-dimensional. The return value `eul` is of class `double`.

Algorithm

`bweuler` computes the Euler number by considering patterns of convexity and concavity in local 2-by-2 neighborhoods. See ¥ for a discussion of the algorithm used.

References


strel

Create morphological structuring element

Syntax

- `SE = strel(shape,parameters)`

Description

`SE = strel(shape,parameters)` creates a structuring element, `SE`, of the type specified by `shape`. This table lists all the supported shapes. Depending on `shape`, `strel` can take additional parameters. See the syntax descriptions that follow for details about creating each type of structuring element.

<table>
<thead>
<tr>
<th>Flat Structuring Elements</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>'arbitrary'</td>
<td>'pair'</td>
</tr>
<tr>
<td>'diamond'</td>
<td>'periodicline'</td>
</tr>
<tr>
<td>'disk'</td>
<td>'rectangle'</td>
</tr>
</tbody>
</table>

60
<table>
<thead>
<tr>
<th>'line'</th>
<th>'square'</th>
</tr>
</thead>
<tbody>
<tr>
<td>'octagon'</td>
<td></td>
</tr>
</tbody>
</table>

SE = strel('diamond', R) creates a flat, diamond-shaped structuring element, where \( R \) specifies the distance from the structuring element origin to the points of the diamond. \( R \) must be a nonnegative integer scalar.

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\( R = 3 \)

SE = strel('disk', R, N) creates a flat, disk-shaped structuring element, where \( R \) specifies the radius. \( R \) must be a nonnegative integer. \( N \) must be 0, 4, 6, or 8. When \( N \) is greater than 0, the disk-shaped structuring element is approximated by a sequence of \( N \) periodic-line structuring elements. When \( N \) equals 0, no approximation is used, and the structuring element members consist of all pixels whose centers are no greater than \( R \) away from the origin. If \( N \) is not specified, the default value is 4.

**Note** Morphological operations run much faster when the structuring element uses approximations (\( N > 0 \)) than when it does not (\( N = 0 \)). However, structuring elements that do not use approximations (\( N = 0 \)) are not suitable for computing granulometries. Sometimes it is necessary for strel to use two extra line structuring elements in the approximation, in which case the number of decomposed structuring elements used is \( N + 2 \).

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\( R = 3 \)

SE = strel('line', LEN, DEG) creates a flat, linear structuring element, where \( LEN \) specifies the length, and \( DEG \) specifies the angle (in degrees) of the line, as measured in a counterclockwise direction from the horizontal axis. \( LEN \) is approximately the distance between the centers of the structuring element members at opposite ends of the line.
SE = strel('octagon',R) creates a flat, octagonal structuring element, where R specifies the distance from the structuring element origin to the sides of the octagon, as measured along the horizontal and vertical axes. R must be a nonnegative multiple of 3.

Notes

For all shapes except 'arbitrary', structuring elements are constructed using a family of techniques known collectively as **structuring element decomposition**. The principle is that dilation by some large structuring elements can be computed faster by dilation with a sequence of smaller structuring elements. For example, dilation by an 11-by-11 square structuring element can be accomplished by dilating first with a 1-by-11 structuring element and then with an 11-by-1 structuring element. This results in a theoretical performance improvement of a factor of 5.5, although in practice the actual performance improvement is somewhat less. Structuring element decompositions used for the 'disk' and 'ball' shapes are approximations; all other decompositions are exact.

Algorithm

The method used to decompose diamond-shaped structuring elements is known as "logarithmic decomposition" [1].

The method used to decompose disk structuring elements is based on the technique called "radial decomposition using periodic lines" [2], [3]. For details, see the MakeDiskStrel subfunction in toolbox/images/images/@strel/strel.m.

The method used to decompose ball structuring elements is the technique called "radial decomposition of sphere" [2].
References


imtophat

Perform top-hat filtering

Syntax

- `IM2 = imtophat(IM,SE)`
- `IM2 = imtophat(IM,NHOOD)`

Description

`IM2 = imtophat(IM,SE)` performs morphological top-hat filtering on the grayscale or binary input image `IM` using the structuring element `SE`, where `SE` is returned by `strel`. `SE` must be a single structuring element object, not an array containing multiple structuring element objects.

`IM2 = imtophat(IM,NHOOD)`, where `NHOOD` is an array of 0's and 1's that specifies the size and shape of the structuring element, is the same as `imtophat(IM,strel(NHOOD))`.

Class Support

`IM` can be numeric or logical and must be nonsparse. The output image `IM2` has the same class as the input image. If the input is binary (logical), the structuring element must be flat.
7. References


13. Mohamed Sammouda, Rachid Sammouda, Noboru Niki, Kiyoshi Mukai. Liver cancer detection system based on the analysis of digitized color images of tissue samples obtained using needle biopsy. Information Visualization. 1.2 (Jun 2002) 130


   <http://rsb.info.nih.gov/ij/plugins/watershed.html>
18. CR Garcia, AR Dluzewski, LH Catalani, R Burting, J Hoyland, and WT Mason.
    Calcium Homestastic in Intraerythrocytic Malaria Parasites. European Journal
    of Cell Biology. 71.4 (December 1996) 409-13
    <http://www.libpng.org/pub/png/pngintro.html>
    <http://rkb.home.cern.ch/rkb/AN16pp/node178.html>
     Chapter5/> 
    Online. 19 April, 2004.
    l#22581>
    <http://sln.fi.edu/biosci/blood/white.html>
    <http://www.drhull.com/EncyMaster/B/blood_cells.html>
    <http://www.mmorph.com/mmtutor1.0/html/mmtutor/mm045residues.html>
    <http://www.poems.msu.edu/InfoMastery/Diagnosis/SensSpec.htm>