Automating Magnetometer Data Processing to Reduce Processing Time and Improve Geophysical Archaeological Survey Results

by

Jeffrey T. Hejna

A senior thesis submitted to the faculty of

the Ithaca College Department of Physics and Astronomy

in partial fulfillment of the requirements for the degree of

Bachelor of Arts

Department of Physics and Astronomy

Ithaca College

May 2016

ITHACA COLLEGE DEPARTMENT APPROVAL

of a senior thesis submitted by

Jeffrey T. Hejna

This thesis has been reviewed by the senior thesis committee and the department chair and has been found to be satisfactory.

Dr. Michael Rogers, Advisor	Date
Dr. Bruce Thompson, Senior Thesis Committee Member	Date
Dr. Kelley Sullivan, Senior Thesis Instructor	Date
Dr. Bruce Thompson, Chair, Department of Physics and Astronomy	Date

I understand that my thesis will become part of the permanent print collection in the Ithaca College Department of Physics and Astronomy Society of Physics student room. A digital copy will also remain on file in the Department. My signature below authorizes the addition of my thesis to this collection.

Jeffrey T. Hejna	Date

THE ABSTRACT OF THE THESIS OF

Jeffrey T. Hejna

for the degree of Bachelor of Arts in Physics

submitted May 2016

Thesis Title:
Automating Magnetometer Data Processing to Reduce Processing Time and Improve Geophysical Archaeological Survey Results

The main objective of this project was to ease the task of processing magnetic data collected by Professor Rogers's geophysical archaeology team. This was accomplished by coding programs using Java that processed the data and eliminated the tedious and time consuming task of copying, pasting, and editing the data manually. I coded two programs to complete the data processing that was originally done by hand. The first program processed the magnetic data of individual units and the second program edge matched these units together to produce an overall subsurface survey map. Upon completing these tasks, a complete subsurface survey map can be used for analysis. These maps are used by archaeologists in order to make deductions about subsurface features and how these features relate to past human interaction with the landscape. Features that have been found from magnetic surveying include Native American pit houses and structures of buildings from ancient cultures. Upon analysis of the images produced from data processing by means of my programs, it was found that my programs were successfully able to process magnetic data. They produce subsurface image maps that are more crisp and contain more magnetic signals which increases the value of these maps in that more features can be analyzed. The programs also streamline the processing of data in such a way that they decrease the amount of time needed to complete a subsurface survey map by a significant margin. If the correct software is present on a laptop, processing of data can be done on site which has significant implications. Problems that arise during the data collection process could be rectified and, if certain features are discovered that were not expected, the surveyors could re-evaluate their survey plans to focus on this new feature.

ACKNOWLEDGMENTS

There are many people I would like to thank, but I would like to start by thanking the Physics and Astronomy department at Ithaca College for the opportunities that were given to me. I would like to specifically thank my research advisor, Dr. Michael "Bodhi" Rogers for the opportunity to work on this project and for his overwhelming patience and support in helping me produce a thesis I could be proud of. I also would like to thank my thesis instructors, Dr. Luke Keller and Dr. Kelley Sullivan for all of their guidance throughout writing this thesis. Lastly, I would like to thank my family, friends, and fellow thesis students for their continued support throughout this expereience, as well as throughout my undergraduate career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The history of magnetism goes back thousands of years with descriptions of magnetism in natural materials first appearing in Chinese literature before AD 400. It wasn't until 1269 when Petrus Pereginus de Maricourt observed the interaction of iron fragments with lodestone specimens that the notion of polarity within magnetic materials was discovered (Aspinall *et al.* , 2008). Adding onto the work done by Petrus, William Gilbert published *De Magnete* in 1600 in which he claimed that the Earth itself was a giant magnet and created its own magnetic field. From the nineteenth century came an abundance of discoveries in regards to magnetism from scientists such as Hans Christian Ørsted who discovered that an electric current was a source of a magnetic field, and André Marie Ampère who proposed the idea of permanent magnets (Aspinall *et al.* , 2008; Brain & Knudsen, 2007). Following all of these discoveries about magnetic fields, Carl Friedrich Gauss successfully measured the strength of these fields in 1832 by using a permanent bar magnet and some gold filament (Aspinall *et al.* , 2008). It is from this experiment that the first practical magnetometer was created.

Since Guass' initial discovery magnetometers were continually developed and enhanced but it wasn't until World War II when there was an incentive to detect sub-

marines that the design and performance of magnetometers would significantly increase (Telford *et al.* , 1990). Magnetometers are now commonly used in the field of geophysical archaeology which involves employing these instruments to measure the changes in the strength of the Earth's magnetic field in order to discover subterranean features. Subsurface features are magnetized by the Earth's magnetic field, thus altering the magnetic field strength. These fluxuations are detected and recorded by magnetometers. The magnetic data (after sufficient processing has been done) allow achaeologists to make deductions about what these subsurface features could be. Two modern day magnetometers that are commonly used for archaeological purposes include the fluxgate magnetometer and the optically pumped cesium magnetometer. Employing these devices allows archaeologists to discover subterranean features that would otherwise be impossible to find without unearthing them by digging. Archaeological excavation provides detailed information about buried features, but it is time consuming, expensive, and alters the site in a non-recoverable way. The fluxgate magnetometer and the optically pumped cesium magnetometer can measure changes in the Earth's magnetic field to 0.1 nT and 0.01 nT, respectively (Tabbagh, 2003). Archaeologists can use the precision of these instruments to make concrete deductions in regards to what kind of features are present and where they are located within the site, without disrupting and altering the site in any way (Aspinall *et al.* , 2008).

The optically pumped cesium magnetometer utilized by Professor Rogers's geophysical archaeology team can take 10 readings every second in continuous mode and measures the magnitude of the magnetic field of the Earth (Rogers, 2011). Surveying large sites results in sizable data sets that need to be processed in order to be used for analysis. Improvements in computers and software are allowing archaeologists to process large data sets collected by magnetometers to further understand how magnetic features relate to past human interaction with the landscape. Gaffney (2000) used a fluxgate magne-

tometer in Wroxeter, Rome and discovered the buried remains of an ancient city, while Rogers (2010) used a cesium magnetometer and discovered pithouses near Silver City, New Mexico. Magnetometers have proven to be a useful tool in the discovery of subsurface features which allows archaeologists to provide greater details about the history of the landscape in which they are found (Becker & Fassbinder, 2015a,b; Byksara *et al.* , 2008; Cammarano *et al.* , 1997; Gerard-Little *et al.* , 2012; Mercer & Schmidt, 2001; Rogers *et al.* , 2006; Schmidt & Fazeli, 2007; Sheriff *et al.* , 2010; Stull *et al.* , 2013, 2014).

Obtaining an accurate image of the subsurface requires the data to be corrected for various errors through post-aquisition processing. Post-acquisition processing involves removing drop outs, de-spiking, de-staggering, de-striping and edge matching, which I will explain in further detail in Chapter 3 (Aspinall *et al.* , 2008). Processing these data in Professor Rogers's geophysical archaeology laboratory at Ithaca College is currently done manually, is time consuming, and is repetitive in a way that lends the processing methods to automation through the use of computer coding. Reducing the amount of time and effort to construct these images is advantageous because the faster the image can be created, the sooner it can be used to find subsurface features that may be important to find or avoid in a survey, depending on the situation at hand. Faster processing also means a reduction in the cost of magnetometer surveys. The survey team would not be needed for a longer duration because the results from the survey would be avaliable for interpretation sooner.

Processing magnetic data in Professor Rogers's laboratory is a time intensive and manually straining task in that it involves obtaining the data from Geometrics, Inc's MagMap2000, copying and pasting data into Microsoft Excel 2010 files, editing them within those files, and obtaining an image of the processed data by using Golden Software's Surfer Version 12 (See Mahoney's (2015) and Woodward's (2015) senior physics theses for detailed instructions on the current methods for processing magnetic data).

This process can take several hours depending on the size of the site. Majority of the time is spent copying, pasting, and editing the data manually which is a highly repetative process. Computer programs can be an incredibly useful tool for solving complex and repetitive problems in a time efficient manner. The goal of my project was to create computer programs that eliminate the tedious process of copying, pasting, and editing the data manually in order to obtain a subsurface map of a site. This process would be accomplished at a quicker rate and would require less physical and mental work by the user than by use of traditional methods. I decided to use Java as my choice of programming language to create the programs which would automate the processing of magnetic data. I made this decision based on the fact that I only know two programming languages thus far; Python and Java. I chose Java over Python because I discovered that Python was much slower than Java when it came to reading the files containing the data. Another factor that influenced my decision was the fact that Java is known for being an object oriented language (Holzner, 2000). By using objects I was able to keep the processed data seperate from the raw data and I could easily access the final data set to be written into a new text file.

The goal of this thesis is to show that by using my computer programs, all previous methods of data processing will be accomplished in less time than by use of traditional methods. The program will also obtain results that are just as good, if not better, than if traditional methods were used. The specific functionality that I set for my program (which will be covered in detail in Chapter 3) include: being able to de-stagger the data of individual units and to edge match adjacent units together.

Chapter 2

Theory

2.1 Induced and Remanent Magnetization

The changes in magnetic field strength that a magnetometer can detect depends on several factors which include, but are not limited to, the location of the site that is being surveyed, how deep the features are under the surface, and the physical properties of the features. This section aims at providing the theory behind the magnetization of materials and objects due to the Earth's magnetic field and how this allows geophysical archaeologists to detect subsurface features.

2.1.1 Induced Magnetization

Induced magnetization occurs when a material is subjected to a magnetic field and the resulting ambient field is enhanced, resulting in the material acting like a magnet (Breiner, 1973). The magnetic moments that exist within the material attempt to become aligned with the external field and thus, the material becomes magnetically polarized (Griffiths, 2012). Some materials are more susceptible to becoming magnetically polarized than others due to their physical composition (see Fig. 2.1).

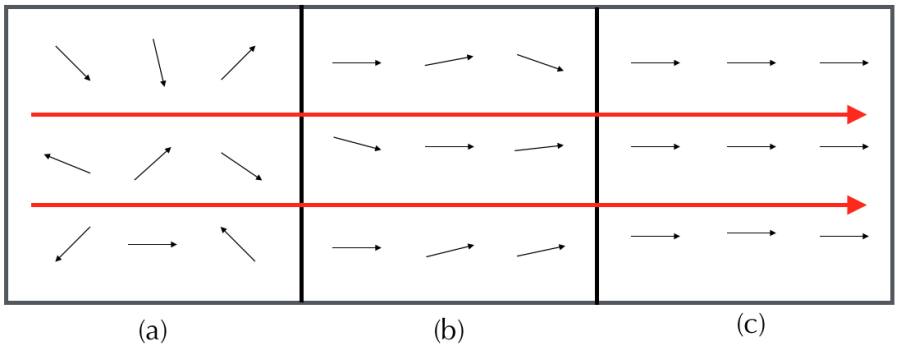


Figure 2.1: When an external magnetic field is introduced (red arrows), the magnetic moments will attempt to align with the field. (a) Materials such as wood don't have moments that are readily influenced by a magnetic field, so there will be a net moment of 0. (b) and (c) could be objects made up of ferrous materials like iron which have magnetic moments that readily align to an external field. (b) shows partial alignment while (c) shows complete alignment.

The magnetization induced in a material, $\vec{M}$, and the external field exposed to the material, $\vec{H}$, have a relationship that can be defined as:

$$\vec{M} = \mu\vec{H}, \tag{2.1}$$

where μ is the material's magnetic susceptibility. $\vec{M}$ and $\vec{H}$ are both measured in Am^{-1} which makes μ dimensionless. For most materials the magnetic susceptibility is less than 1 but for ferromagnetic materials like iron, it can be between 1 and 1,000,000 (Breiner, 1973).

The induced magentic field, $\vec{B}$, and $\vec{H}$ are closely related and the relationship between them is given by:

$$\vec{B} = \mu_0(\vec{H} + \vec{M}), \tag{2.2}$$

where μ_0 represents the magnetic permeability which has a value of $4\pi \times 10^{-7}\,\mathrm{TmA}^{-1}$.

By plugging Eqn. 2.1 into Eqn. 2.2, the following result is obtained:

$$\vec{B} = \mu_0\vec{H}(1 + \mu), \tag{2.3}$$

where the quantity $(1+\mu)$ represents the relative permeability of the medium that the magnetic field travels through. Thus, it stands to reason that materials that are made up of highly ferrous metals such as iron will have a large overall flux density when exposed to an external magnetic field.

Features in the subsurface that contain magnetic minerals like iron will experience magnetic induction due to the Earth's magnetic field and the resulting magnetization of these objects increases the strength of the total magnetic field, as evidenced by Eqn. 2.3. It is important to note that magnetic field strength drops off with distance so magnetometers can only record the magnetic signals associated with magnetic objects that are often within the first few meters beneath the ground surface (Aspinall *et al.* , 2008).

2.1.2 Remanent Magnetization

Remanent magnetization is similar to induced magnetization in that the same logic is applied from the previous section; the Earth's magnetic field subjects the moments within the material to become aligned with the field. The key difference between induced and remanent magnetization is that remanent magnetization occurs when a material that contains magnetic moments has been heated above their Curie temperatures such that the moments readily align with the Earth's magnetic field at that time (Aspinall *et al.* , 2008). After the material has cooled the moments remain aligned in the direction that the field was pointing in at the time. This is an important aspect because the remanent magnetization of an object may or may not be in the same direction as the earth's magnetic field presently which has ramifications when interpreting the data collected by the magnetometer (Breiner, 1973).

2.2 Coding Implementation

There are limitless possibilites to solving a problem in regards to computer programming and although there may be different solutions to the same result, time and computer memory efficiency are the key to a pratical solution. My programs need to be able to process large amounts of data collected by the magnetometer which is why, by carefully considering the implications of the various data structures I had at my disposal, I decided to use both arrays and ArrayList to minimize the amount of time and memory usage my programs would take to process the data. The following sections will elaborate specifically on the theory behind the time and memory effiiency of arrays and ArrayList.

2.2.1 Arrays

Arrays are an incredibly useful way to store multiple data for easy access and they are a fundamental data structure within Java (Holzner, 2000). They can be thought of as a list that has a set length and can only hold the same type of information within them. Arrays are also advantageous because by assigning a numeric value to each element within the array, it becomes very easy to increment the array index and access all elements within the array (Holzner, 2000).

The attribute of arrays that brings up concern when programming is the fact that their size cannot be changed. Java only allocates the amount of memory designated by the size the programmer has requested when an array is initialized (Holzner, 2000). Using arrays is advantageous when the size is always known because then it is possible to reduce the amount of memory used. The amount of data collected by the mangnetometer fluctuates constantly so it is impossible to know the correct size to make the arrays without counting the amount of lines which would take more time. The constraint of size with arrays is why it is necessary to consider using a different data structure in Java: ArrayList.

2.2.2 ArrayList vs. LinkedList

ArrayList is a special kind of data structure in that it uses arrays. A key difference between arrays and ArrayList is that ArrayList is not constrained to a set size. As previously stated, the amount of data to be processed can vary significantly so the use of ArrayList is advantageous because even if the array fills up, ArrayList will automatically create a new array that has a capacity of 1.5 times the original and copy all the information from the old array to this new one. This does take time and more memory to accomplish, but the trade off is far better than having the program crash from trying to use an array and not having enough space.

In order to gauge the time effieciency of ArrayList it is necessary to compare it to another type of list, specifically LinkedList. LinkedList is different in that it uses nodes to store data instead of arrays. A node can be thought of as a space in memory that holds onto two pieces of information: the data that are put there, and a pointer to a memory location of another node, if there is one. These nodes are then 'linked' together in that they point to each other which makes it possible to access all the elements contained within the list. LinkedList has the same methods as ArrayList, meaning that it is possible to add items, remove items, and get the data stored at a certain node (Holzner, 2000). What is interesting is that the time efficiency of calling the methods of ArrayList and LinkedList varies drastically.

The main methods used within my programs include adding elements to a list and getting the information from the lists, so by comparing the add() and get() methods of ArrayList and LinkedList, it is possible to determine the more time efficient option. Fig. 2.2 represents the mathematical trends for adding and getting items from ArrayList and LinkedList. O(1) represents the idea that no matter how many items are in the list, it only takes one additional step to accomplish a task. Methods that take only one step to accomplish include the get() method in ArrayList, the add() method to the end of

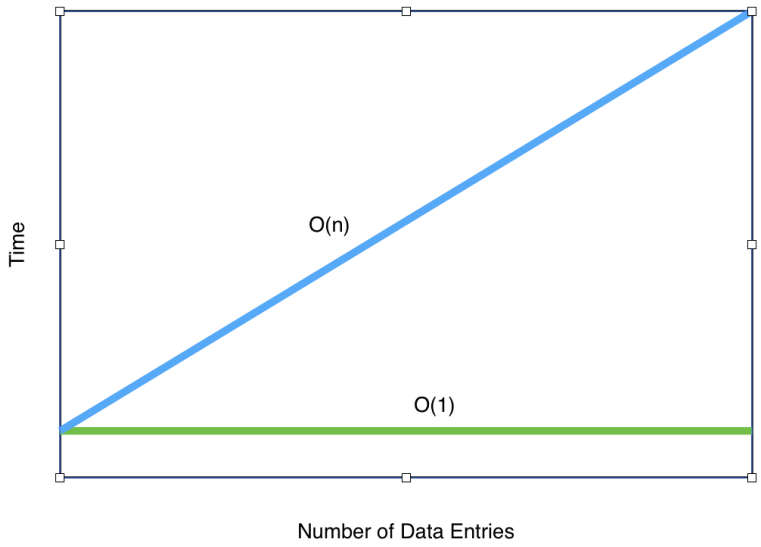


Figure 2.2: These are the mathematical models that represent the amount of time methods of certain data structures take, with some data structures being more time efficient than other. O(n) represents the notion that as the number of elements in a list increases, the time to accomplish a task increases. O(1) represents the idea that as the number of elements in a list increases, the time to accomplish a task remains constant.

an ArrayList, and the add() method to anywhere in a LinkedList. O(n) represents the idea that to accomplish a task, it takes a constantly increasing time as the size of the list increases. Tasks that take this approach include the get() method in LinkedList and adding an element to the beginning of an ArrayList (Program Creek, 2016). The fact that LinkedList does not have the ability to give numeric values to each element is why ArrayList is much faster at the get() method. Likewise, the fact that ArrayList would have to shift each element over one to add an element to the beginning, LinkedList is able to switch their pointers so it is much faster.

When looking at memory usage between ArrayList and LinkedList it is worth noting that nodes have to store both the data and the memory of the next element while ArrayList elements only hold onto the element itself. Fig. 2.3 contains information pertaining to the memory usage of a LinkedList versus an ArrayList as the number of elements within them increases. It is revealed that as the number of elements increases,

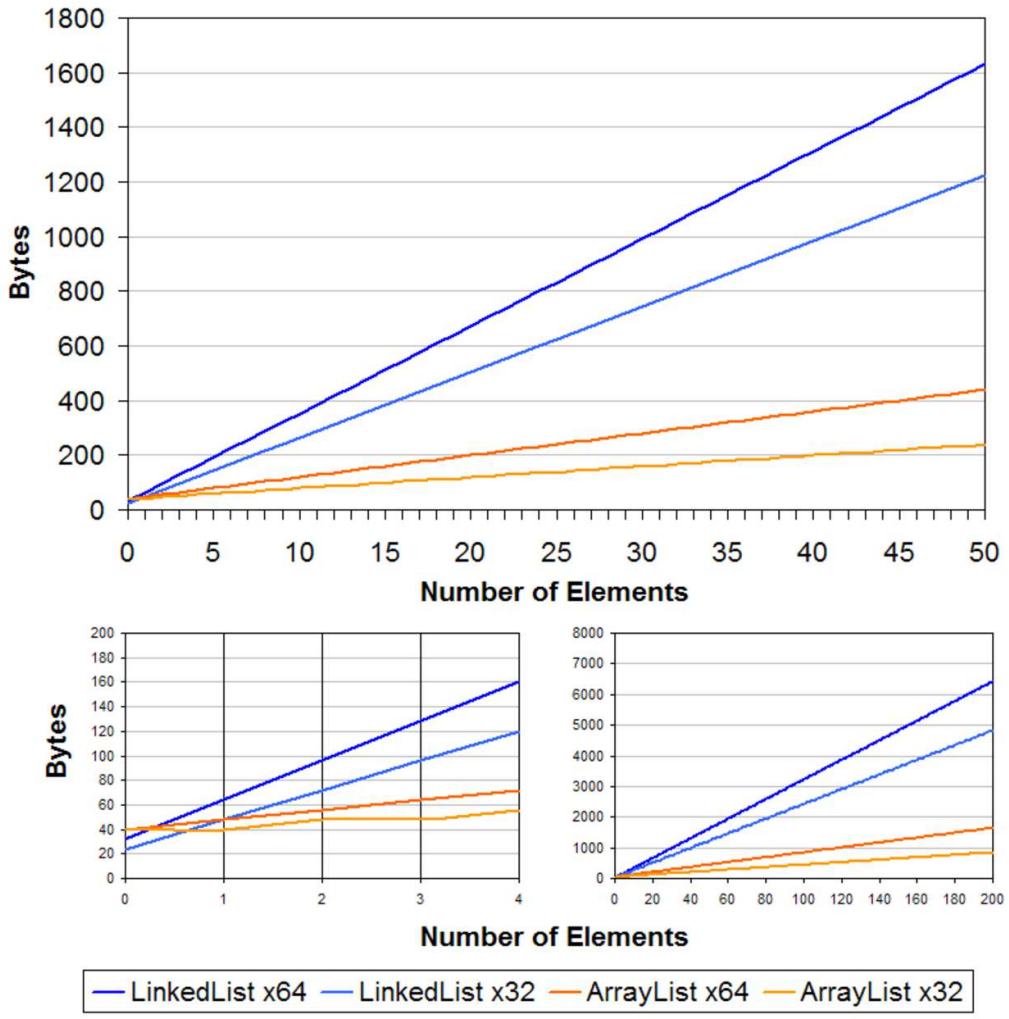


Figure 2.3: As the number of elements increases in an ArrayList and LinkedList, the amount of memory that is being used increases for both. However, LinkedList increases at a much faster rate than ArrayList. This has significant implications in that using more memory slows down the programs.

LinkedList uses much more memory than ArrayList (Stack Overflow, 2016).

Choosing time and memory efficient data structures is the key to optimizing the overall quality of a computer program. ArrayList is faster at getting elements than LinkedList which will allow my programs to run much faster. ArrayList also uses less memory compared to LinkedList when there is a lot of data to be stored. It makes more sense to use ArrayList than LinkedList since there are thousands of data values to be stored from the magnetometer.

Chapter 3

Methods

3.1 Processing Data: Traditional Methods

In order to process the raw data collected by the magnetometer they need to be uploaded from the magnetometer to the computer by utilizing the import feature in MagMap2000 (Geometrics, 2002). After the upload is complete there are multiple *.BIN files with each of them correlating to a specific unit from a survey. By utilizing Geometrics, Inc's MagMap2000 (MagMap), Microsoft Excel 2010 (Excel), and Golden Software's Surfer Version 12 (Surfer), the magnetic data can be processed to correct for positional errors, removing drop outs, de-spiking, de-striping, gridding, de-staggering, and edge matching. The following sections will describe these processes and how they have been accomplished by use of traditional methods in order to obtain a complete subterranean map.

3.1.1 Processing in MapMap

Each unit is saved as a *.STN file upon uploading the data from the magnetometer. These files must be opened in MagMap to begin initial processing. Fixing positional errors, removing dropouts, de-spiking, and de-striping must be done in MagMap before

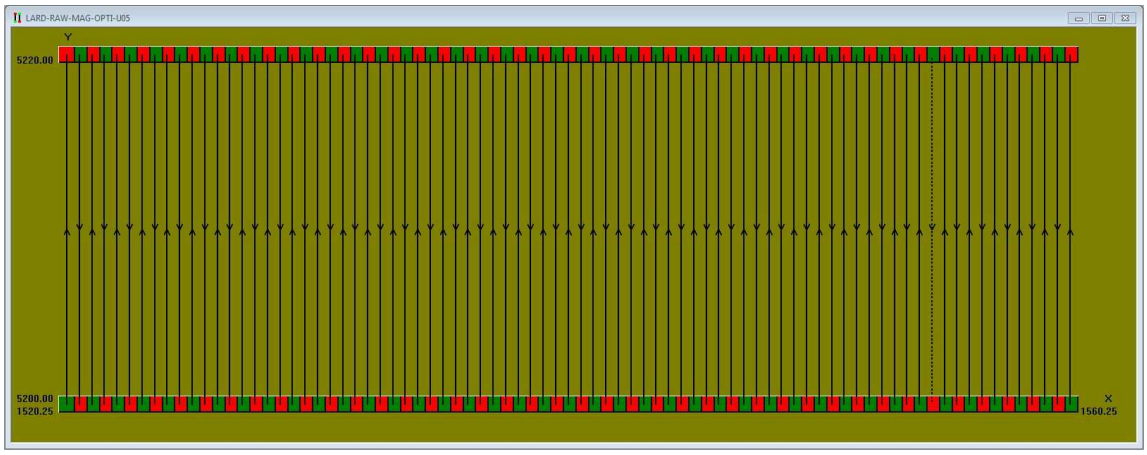


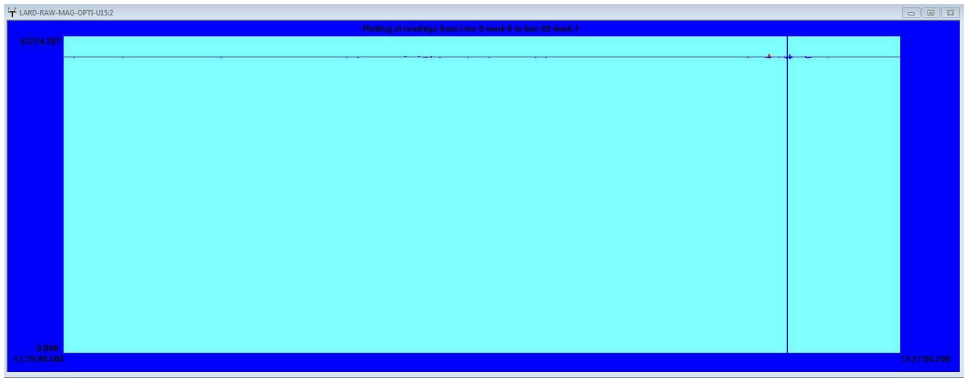
Figure 3.1: This is what is presented when a *.STN file of a unit is opened in MagMap. The lines represent the transects of a unit where data were taken by the magnetometer. Arrows represent the direction in which the magnetometer was traveling. One of the lines is dashed which means that something went wrong with the data collection during that transect.

exporting the data to a *.DAT file. The following sections explain these processes in greater detail.

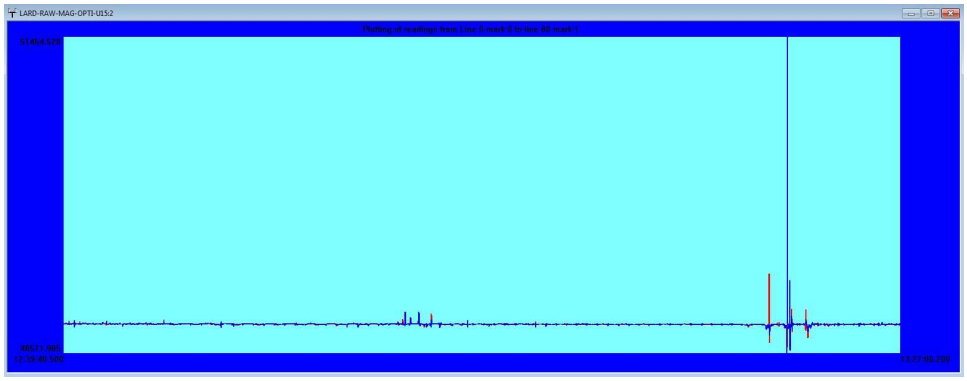
Fixing Positional Errors

While surveying, the operator of the magnetometer keeps the left wheel of the non-magnetic cart on the transect line, which places the center of the cart 0.25 m off the line. It is imperative to shift the entire unit by 0.25 cm horizontally in order to account for the shift of position of the sensors on the magnetometer. This can be corrected by right clicking on the opened *.STN file for the unit (see Fig. 3.1), selecting **Shift Grid**, and typing in 0.25 next to **X**.

In some cases there is a need to remove a line completely from a unit. This needs to be done when errors occur within the magnetometer (see Fig. 3.1). In order to remove a faulty line one has to right click the specific line, select **Delete Segment**, and click **Yes**.



(a)

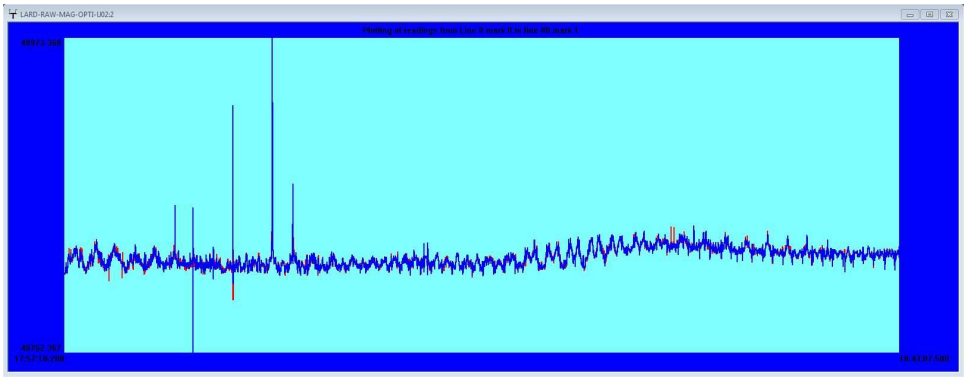


(b)

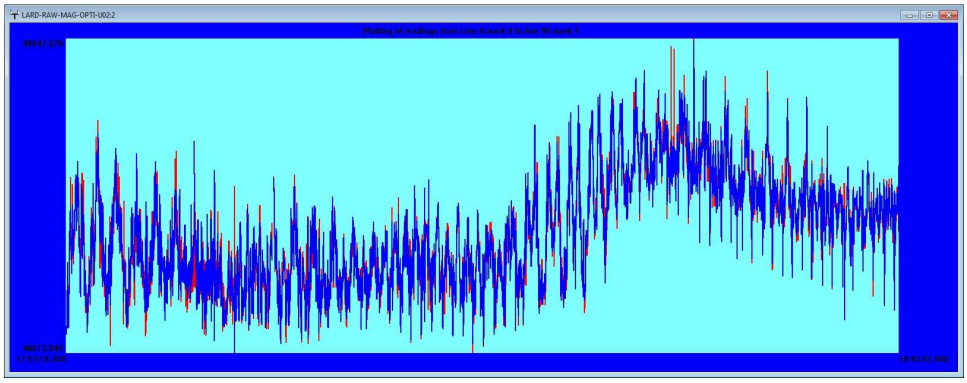
Figure 3.2: The magnetic profile of a particular unit before and after removing drop outs. (a) Before processing is done there may be some drop outs within the data which correspond to a magnetic reading of 0 nT. (b) These drop outs need to be removed to ensure that the magnetic signals aren't drowned out. Although there are no drop outs in (b), there are still large spikes that need to be removed.

Removing Drop Outs

While taking measurements the magnetometer sometimes fails to obtain a reading. In this case the magnetometer will input an empty string for the reading at that instance (Woodward, 2015). Fig. 3.2 shows what the data looks like before and after removing drop outs. To remove drop outs one has to select the **Filter** option and then select **Remove Drop Outs**.



(a)

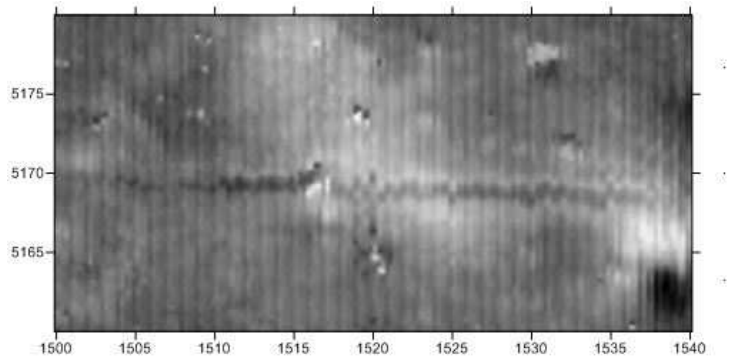


(b)

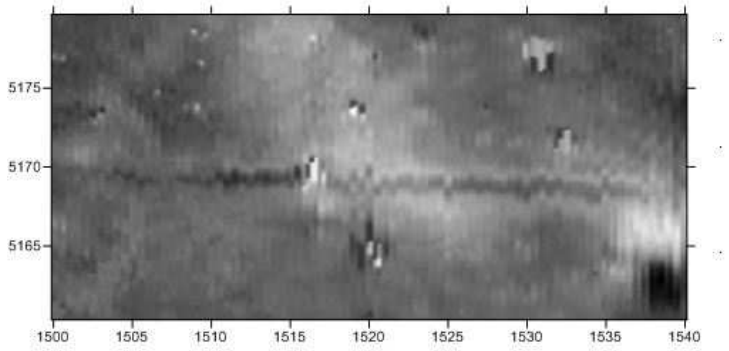
Figure 3.3: Spikes in magnetic data are caused when a highly ferrous material is at or near the surface. (a) Spikes tend to stifle other readings which is why it is important to remove them. (b) The magnetic signals are more prominant after spikes are removed from the data.

De-Spiking

One of the most common errors to remove from magnetometer data sets are random spikes that result from the magnetometer detecting a highly ferrous object (Aspinall *et al.* , 2008). It is advantageous to remove these spikes because they can stifle other readings that may hold valuable information. After de-spiking magnetic data the profile fleshes out and more readings can be seen (see Fig. 3.3). Refer to Colleen Mahoney's (2015) senior thesis for detailed steps on how to de-spike data.



(a)



(b)

Figure 3.4: Striping can occur when surveying in a bidirectional manner. (a) represents a unit that has significant striping. (b) is obtained after removing stripes which improves the clarity of the image.

De-Striping

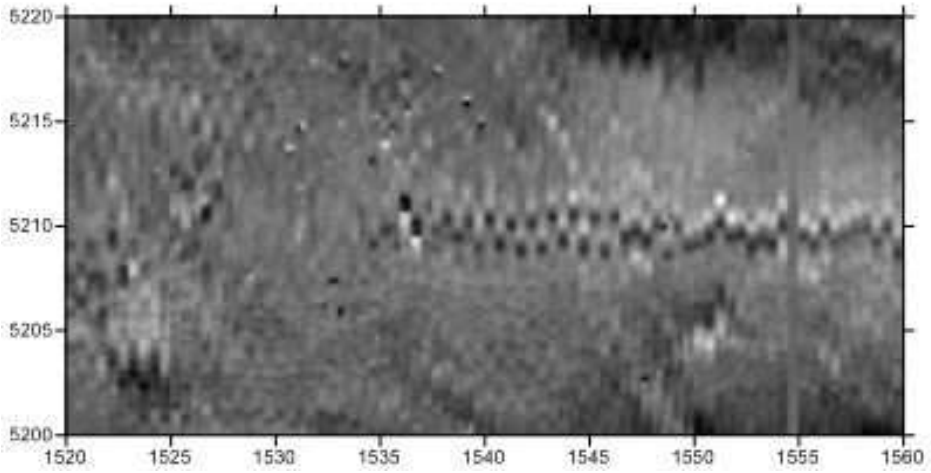
Striping occurs when data are gathered using a bidirectional survey method that places the sensors in the operator's north pole when following a line in one direction, and are in the operator's south pole when traveling along the following line in the opposite direction (Mahoney, 2015). The grid will appear to have a pattern of stripes as a result (see Fig.3.4(a)). Striping doesn't always happen so only grids that have been affected by striping need to be de-striped. Refer to Mahoney (2015) for detailed steps on de-striping data. Fig. 3.4(b) is obtained after de-striping the unit presented in Fig. 3.4(a) .

3.1.2 Gridding in Surfer

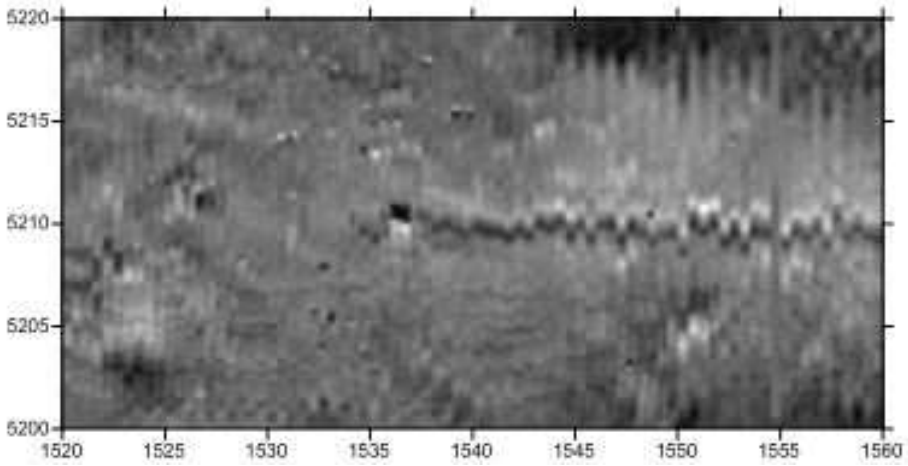
The *.DAT files used will most likely be irregularly spaced due to the fact that the operator may not have maintained a constant pace while surveying (Woodward, 2015). Irregularly spaced means that there are points missing within a plane of points which don't follow any particular pattern (Software, 2009). Surfer provides various gridding methods which constructs an array of regularly spaced values from the previously irregularly spaced ones (Software, 2009). Surfer has different statistical methods to grid data, but I will focus on kriging which is the most commonly used method in Professor Rogers's laboratory. Kriging is a statistical method for interpolating spacial data using linear prediction (Stein, 2012). This process interpolates and adds points between the transects yielding an image of higher resolution as a result (Woodward, 2015). Refer to Mahoney's (2015) thesis for detailed steps on how to grid in Surfer.

3.1.3 De-Staggering

There is a large dependence on the operator maintaining a constant pace as a unit is being surveyed. If the operator's pace changes significantly from line to line the grid will appear to be staggered which could be compared to a zipper (see Fig. 3.5(a)). Excel is needed to edit the data to properly de-stagger a unit. By selecting the odd lines and shifting the Y-coordinates up by a small increment (usually between 0.1 cm and 0.7 cm) and shifting the even lines down the same increment, the lines in the unit have a better chance of lining up correctly (see Fig. 3.5(b)). This is a trial and error based process because it may take several attempts to obtain an image that is the least staggered. De-staggering is only necessary if there is a lot of staggering throughout the unit. This step can be avoided all together if the operator can maintain a constant pace while surveying.

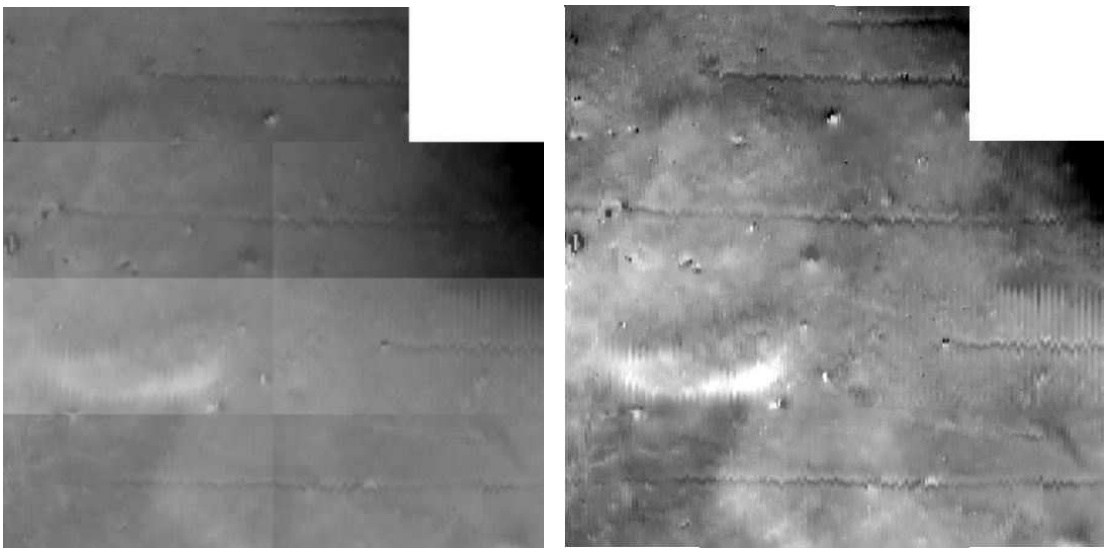


(a)



(b)

Figure 3.5: (a) Staggering can occur while surveying if a constant pace is not maintained by the operator. De-Staggering units requires shifting the data by editing them in Excel sheets. (b) Upon completion of the edits, the unit doesn't appear to be staggered anymore.



(a) (b)

Figure 3.6: (a) Individual units that have been put together before Edge Matching will have distinct edges between them. (b) Edge matching is accomplished by editing the data in Excel sheets until there are no discontinuities at the edges of adjacent grids.

3.1.4 Edge Matching

After processing each unit in MagMap it will need to be merged with others to form the final image of the entire survey. While adding the grids together there will most likely be distinct edges between neighboring grids (see Fig. 3.6(a)). These edges appear due to fact that the Earth's magnetic field does not remain constant and units are surveyed at different times of the day. Detailed information on edge matching can be found in Woodward's (2015) thesis. This process requires the user to copy data from neighboring units into an Excel file and making sure the copied data from each unit lines up with the correct edge that needs to be matched. With larger sites there will be many edges that need to be matched which can result in a lot of time being spent edge matching. Once this process has been completed for all the units the edges between them disappear and the entire survey image becomes much clearer than before (see Fig. 3.6(b)).

3.2 Creating the Code

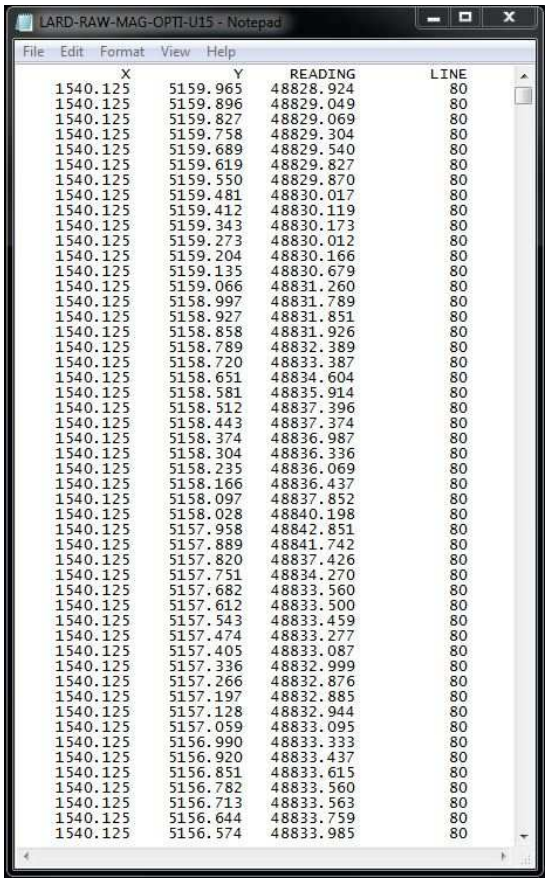
The methods used in the laboratory today are tedious and repetitive which can be replaced by the use of a computer program. I created two different programs with each of them accomplishing different tasks. The first program processes units for the errors listed above with the exception of de-spiking which is still done in MagMap and gridding which is done in Surfer. The second program performs the operations required for edge matching but Surfer is once again still used for gridding. The following sections will discuss how the code was written and the logic behind it.

3.2.1 Reading the *.DAT file, Shifting Data, and Removing Dropouts

After the initial processing done in Magmap each file is then exported as a *.DAT file (see Fig. 3.7). It was advantageous to create a graphical user interface (GUI) in my getFile() function to allow the user to select the appropriate file from the computer that needed to be processed (see Fig. 3.8).

Taking the data presented in the *.DAT file and reading them into my program required the use of arrays. In the readFile() function I was able to skip the first line of the file which contained the column headers and then go through each subsequent line and add each of the four values to the appropriate array of X values, Y values, Readings, and Lines. Having separate arrays was paramount because I was able to edit all of the arrays at once or I could edit them individually while keeping the others the same. With each of the X values I had to add 0.25 in order to account for the position of the sensors on the magnetometer.

Since the magnetometer inputs a blank string when no reading is collected I coded a condition that if it found a reading that contained a blank string, it would set that



X	Y	READING	LINE
1540.125	5159.965	48828.924	80
1540.125	5159.896	48829.049	80
1540.125	5159.827	48829.069	80
1540.125	5159.758	48829.304	80
1540.125	5159.689	48829.540	80
1540.125	5159.619	48829.827	80
1540.125	5159.550	48829.870	80
1540.125	5159.481	48830.017	80
1540.125	5159.412	48830.119	80
1540.125	5159.343	48830.173	80
1540.125	5159.273	48830.012	80
1540.125	5159.204	48830.166	80
1540.125	5159.135	48830.679	80
1540.125	5159.066	48831.260	80
1540.125	5158.997	48831.789	80
1540.125	5158.927	48831.851	80
1540.125	5158.858	48831.926	80
1540.125	5158.789	48832.389	80
1540.125	5158.720	48833.387	80
1540.125	5158.651	48834.604	80
1540.125	5158.581	48835.914	80
1540.125	5158.512	48837.396	80
1540.125	5158.443	48837.374	80
1540.125	5158.374	48836.987	80
1540.125	5158.304	48836.336	80
1540.125	5158.235	48836.069	80
1540.125	5158.166	48836.437	80
1540.125	5158.097	48837.852	80
1540.125	5158.028	48840.198	80
1540.125	5157.958	48842.851	80
1540.125	5157.889	48841.742	80
1540.125	5157.820	48837.426	80
1540.125	5157.751	48834.270	80
1540.125	5157.682	48833.560	80
1540.125	5157.612	48833.500	80
1540.125	5157.543	48833.459	80
1540.125	5157.474	48833.277	80
1540.125	5157.405	48833.087	80
1540.125	5157.336	48832.999	80
1540.125	5157.266	48832.876	80
1540.125	5157.197	48832.885	80
1540.125	5157.128	48832.944	80
1540.125	5157.059	48833.095	80
1540.125	5156.990	48833.333	80
1540.125	5156.920	48833.437	80
1540.125	5156.851	48833.615	80
1540.125	5156.782	48833.560	80
1540.125	5156.713	48833.563	80
1540.125	5156.644	48833.759	80
1540.125	5156.574	48833.985	80

Figure 3.7: A sample *.DAT file from a grid which has four separate columns of data: X, Y, Reading, and Line. These text files usually have upwards of 30,000 lines of data.

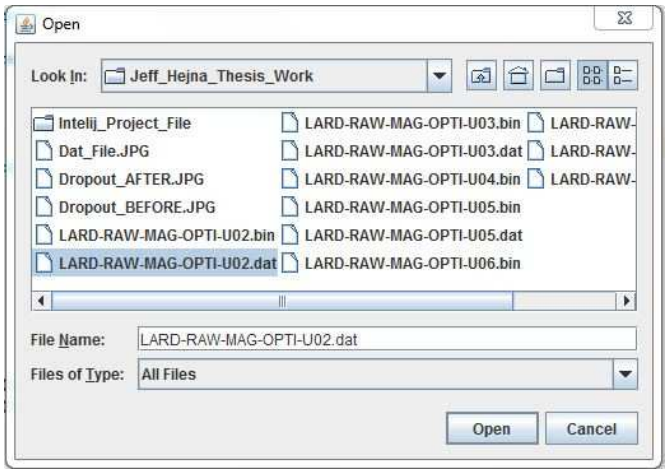


Figure 3.8: This is the GUI used to select the appropriate *.DAT file from a location on the computer. The file selected is the one that is read and processed by my program.

reading to 0.0. Then I had the computer go through all of the data and add them to new arrays only if they didn't have a reading of 0.0, thus removing dropouts. These new arrays of X values, Y values, Readings, and Lines were now clear of dropouts and all the X values were shifted accordingly.

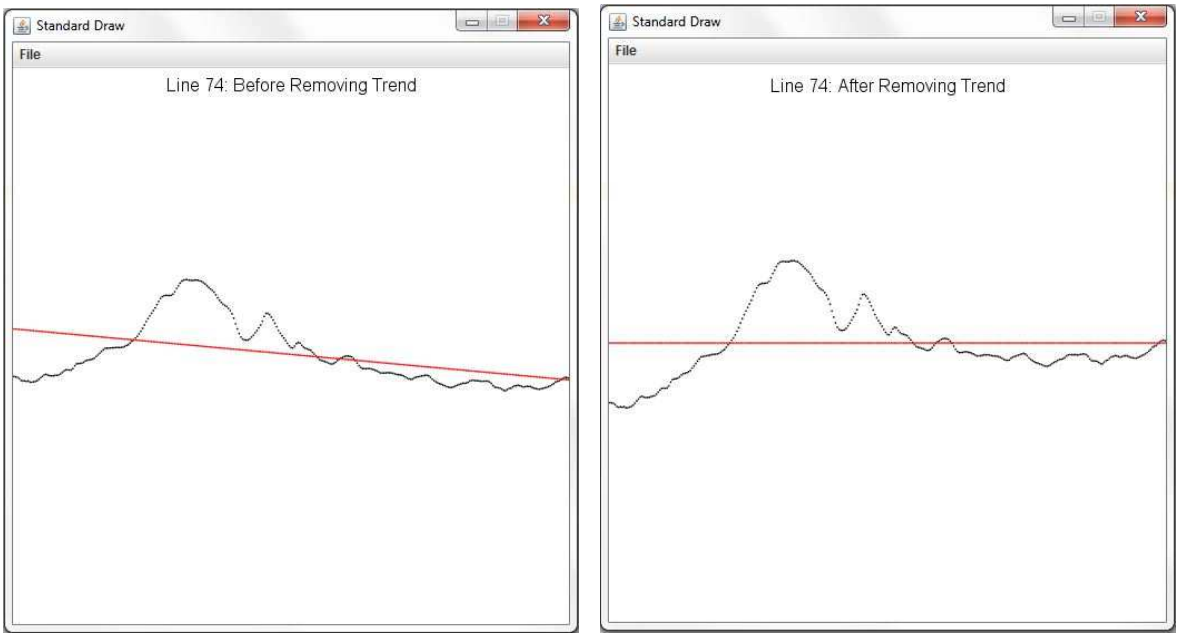
Each sensor has its own data which is why I split up the readings collected by the left sensor and the ones collected by the right sensor. By splitting up the data into two seperate arrays (one for the left sensor and the other for the right sensor) I was able to process the left sensor's data first and the right sensor's data second.

3.2.2 De-Striping

Each line number corresponds to a line where readings were taken from the magnetometer. The number of lines depends on the distance between consecutive lines and the size of the unit being surveyed. For each sensor I had the computer go through the arrays and pick out data corresponding to a specific line number in order to obtain the trend of the magnetic field for that particular line (see Fig. 3.9(a)). A best fit line was also added and was calculated in my calcRegression() function by utilizing a least-squares regression algorithm.

Professor Rogers's and I hypothesized that if my program could subtract the regression line values from the data values that this would remove striping from our data while still preserving the overall characteristics of the data. This approach, otherwise known as the Zero Mean Line method, would cause all of the trends to have a best fit line with zero slope, thus causing all of the data to be centrated on this line while still preserving its characteristic signals (see Fig. 3.9(b)).

After accomplishing this task I then added the X values, Y values, Readings, and Line numbers to the finalXValues, finalYValues, finalReadings, and finalLines ArrayLists that I created as private data members. These ArrayLists would be hold the proccessed data



(a) (b)

Figure 3.9: (a) The magnetic trend of each line can be graphed and a best fit line can be calculated for these data. (b) Upon applying the Zero Mean Line method, the best fit line is subtracted from the data, yielding a graph with similar characteristics, but has a best fit slope of zero.

that would be used to create the final *.DAT files to be used for gridding. This adding to the ArrayLists would happen for each line and for each sensor, thus ensuring that all of the data are processed for errors with the exception of staggering which may or may not be present.

3.2.3 De-Staggering and Creating New *.DAT Files

De-Staggering within my program is similar to the traditional method in that it takes the data from odd lines and shifts those Y values up an amount and takes the data from even lines and shifts those Y values down. The amount they get shifted by is in the range of 0.1cm and 0.7 cm since this is the range that is normally used if staggering occurs at all. My program creates several files with one of them not being staggered at all while the

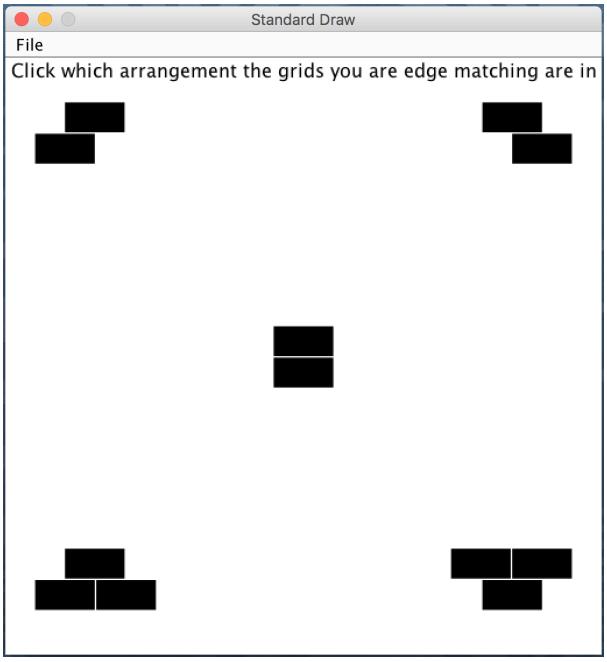


Figure 3.10: When the user wants to edge match adjacent units, the following interactive graphic will be displayed. The user has to click on the orientation that fits with the units they are working with.

other seven files are each shifted an amount between 0.1 cm and 0.7 cm. Multiple files will be avaliable for the user to try in order to find the one that is the least staggered. The user can then delete the other files that weren't used and the file they chose will be the one that is used in Surfer for gridding and for edge matching.

3.2.4 Edge Matching

After processing, the units need to be edge matched together to produce the final subsurface map. I created an interactive layout that allows the user to select which orientation the units are in since adjacent units may be in different orientations with respect to each other (see Fig. 3.10). Units may be next to each other horizontally which would originally require edge matching but this is not necessary due to the way my program originally processed each unit. This will be discussed in greater detail in Chapter 4.

The program asks the user what files are to be edge matched after the orientation has been chosen. These files will be 'combined' in a sense that my program will create a new file that has the coordinates of the two grids together. By following the basic algorithm for edge matching provided by Woodward (2015) I was able to have the computer compute the values that would be added to the new file (*.TIP). This file is then gridded in Surfer and by adding this gridded *.TIP file to the other grids, the edge between the units is eliminated.

3.3 Output: Program vs. Traditional Methods

It was necessary to compare the time it took to create a survey image using both methods in order to determine which method was faster and it was also necessary to compare these images to determine if my program created them with higher quality. I will be talking about two different sites that were surveyed by Professor Rogers's geophysical archaeology team in order to determine the quickness of my program to process these data as well as its ability to create higher quality survey maps. The surveys completed at the Larder Site in Henderson, Nevada and at the Seward House Museum National Historic Landmark in Auburn, New York were processed using traditional methods as well as by using my programs and the resulting images were then compared by Professor Rogers and myself in order to determine whether my program created higher quality survey images. I will talk generally about the difference in time it took to produce these images in the following chapter since I processed all of these data by myself.

3.4 Testing Usability

Testing to see if my program is user-friendly is important in that this program will be used for future projects and if the program is difficult to use, then that will deter people

from using it. I had four participants use my programs while following the instructions I provided to access whether the instructions were clear and whether my programs are easy to use or not (see Appendix A for Instructions).

For each participant I sat with them and while they used the program I took notes and answered questions if any were raised. The participants were asked to process two adjacent units and then edge match them together. Upon completing the task, the participant was given a questionaire that asked the following questions:

1. On a scale from 1-10, with 1 being very difficult and 10 being very easy, rate how difficult it was to process an individual unit. Explain your choice.

2. On a scale from 1-10, with 1 being very difficult and 10 being very easy, rate how difficult it was to edge match units. Explain your choice.

3. On a scale from 1-10, with 1 being very confusing and 10 being very clear, rate the clarity of the instructions. Explain your choice.

4. List anything else you would like to be addressed to improve the quality of the instructions and/or my program:

The ratings and suggestions helped determine how easy the programs are to operate and helped solidify any issues that came up in regards to the language of the instructions.

Chapter 4

Survey Image Creation and Analysis

4.1 Raw Survey Data

Typical magnetic readings collected by Professor Rogers's geophysical archaeology team are around 50,000 ± 2,000 nanotesla (nT). These readings depend a mulititude of variables including where the site is located in the world as well as the kind of environment being surveyed. Surveys done in open fields had readings around 48,000 nT while sites like cemetaries averaged at roughly 52,000 nT. Sites where there is significant human presence tend to have a large amount of ferrous material in them, which can explain these larger readings (Aspinall *et al.* , 2008). This is due to the fact that sites like cemetaries typically have many ferrous objects like fences and caskets which infuences the magnetic readings in the positive direction.

Dropouts in the data are recognizable in that the reading will be 0 nT. Spikes that are present are similar to dropouts in that they are recognizable because they are typically several thousand nT above the average readings. The readings at the Larder site averaged at roughly 48,800 nT while the data at Seward House averaged around 52,600 nT. These sites and the processing of their data will be the discussion of the rest of this chapter.

4.2 Larder Site, Clark County Wetlands, Henderson, NV

The survey that took place in Henderson, Nevada was processed by myself using both traditional methods and my program. The data that was processed using traditional methods can be seen in Fig. 4.1 while the data that was processed using my program can be seen in Fig. 4.2. It took well over 10 hours to process the data using traditional methods. In this data set, there were 16 individual grids to be processed and then edge matched together. This resulted in many hours of copying and pasting data, which resulted in myself feeling mentally fatigued and contributed to mistakes being made. Traditional methods also require grids that are horizontally adjacent to be edge matched which is taken care of automatically by using my program. Professor Rogers and I hypothesized that by using the Zero Mean Line method, the magentic trends from horizontally adjacent units line up such that there are no distinct edges between them. By using my programs I was able to process all the data and produce Fig. 4.2 in less than 2 hours. This has significant implications in that if all of the needed software is present on a laptop, data could be processed on site with ease and any significant errors could be exposed, allowing for data to be retaken.

Professor Rogers and I compared Figs. 4.1 and 4.2 and determined that my program created a higher quality image in that the features of the image are more crisp and distinct than those in the image created from traditional methods. This is advantageous because not only will my program process data in a shorter amount of time, but it will result in higher quality survey maps to be used in finding subsurface features.

4.3 Seward House National Historic Landmark, Auburn, NY

The magnetic data collected from the Seward House Historic Landmark in Auburn, NY was processed by myself using both traditional methods as well as my program and the resulting images can be seen in Figs. 4.3 and 4.4. Professor Rogers and I analyzed the two images and found that the data processed by my program produced an image with greater quality than the one processed by traditional methods. Fig. 4.3 has a blackend edge on the left while Fig. 4.4 does not. This blackend edge corresponds to a metalic fence which caused significant noise in the magnetic readings. While the traditional methods were unsucessful in removing this noise, my program was able to remove it and reveal signals along this edge that are not seen in Fig. 4.3.

The amount of time needed to process these data with my program was again less than the time needed using traditional methods. The time taken to process the data using traditional methods was about two hours, while the use of my program allowed me to accomplish this task in about 30 minutes. This reduction of time shows once again that my program can process these data at a much faster rate than by use of traditional methods and that the results are more crisp and, therefore, more useful in analysis of the types of subsurface features present.

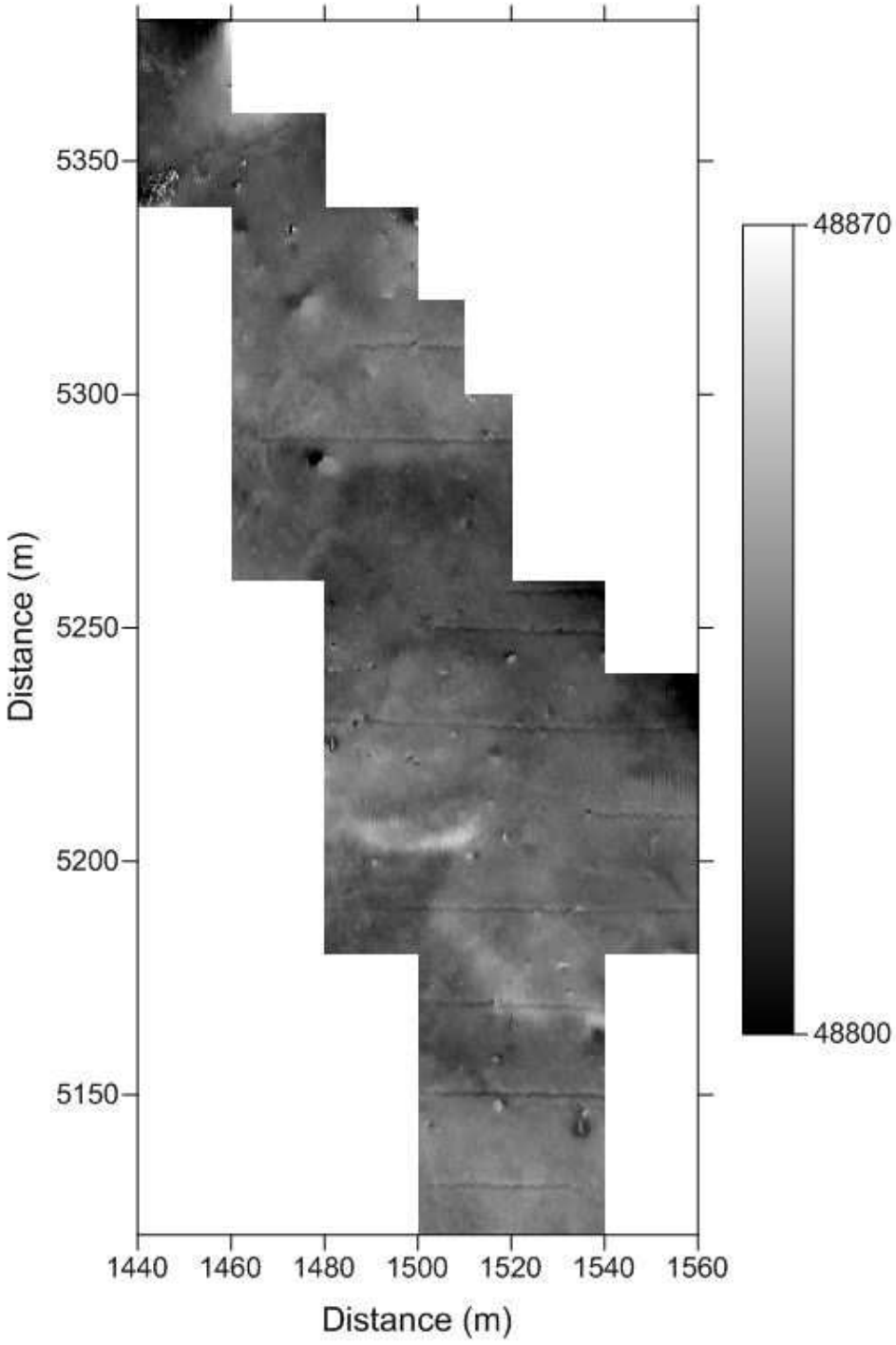


Figure 4.1: The final subsurface survey map of the magnetic data collected at the Larder Site in Henderson, NV. These data were processed by use of traditional methods which took well over 10 hours to accomplish.

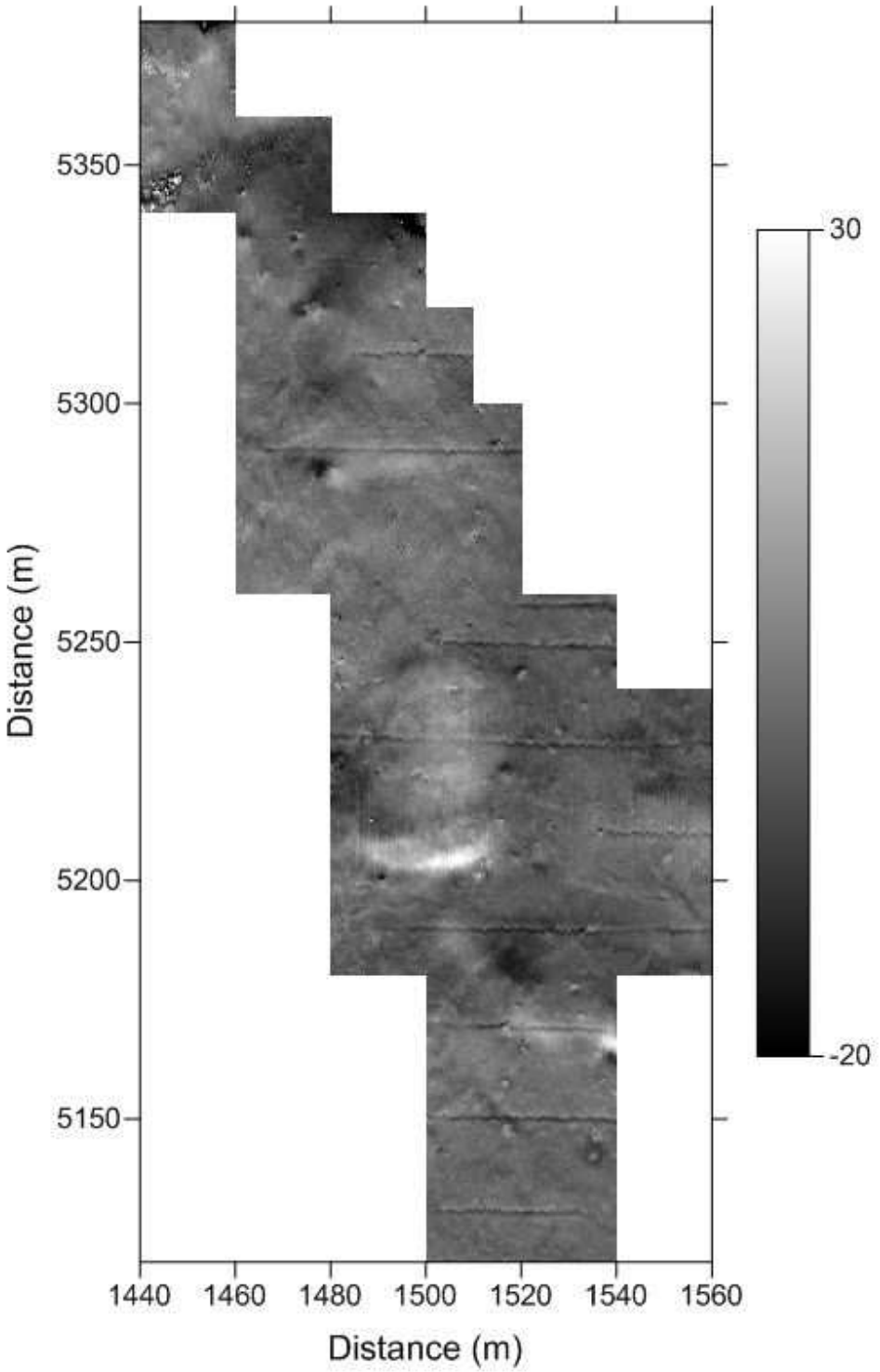


Figure 4.2: The final subsurface survey map of the magnetic data collected at the Larder Site in Henderson, NV. These data were processed by use of my programs in less than 2 hours time.

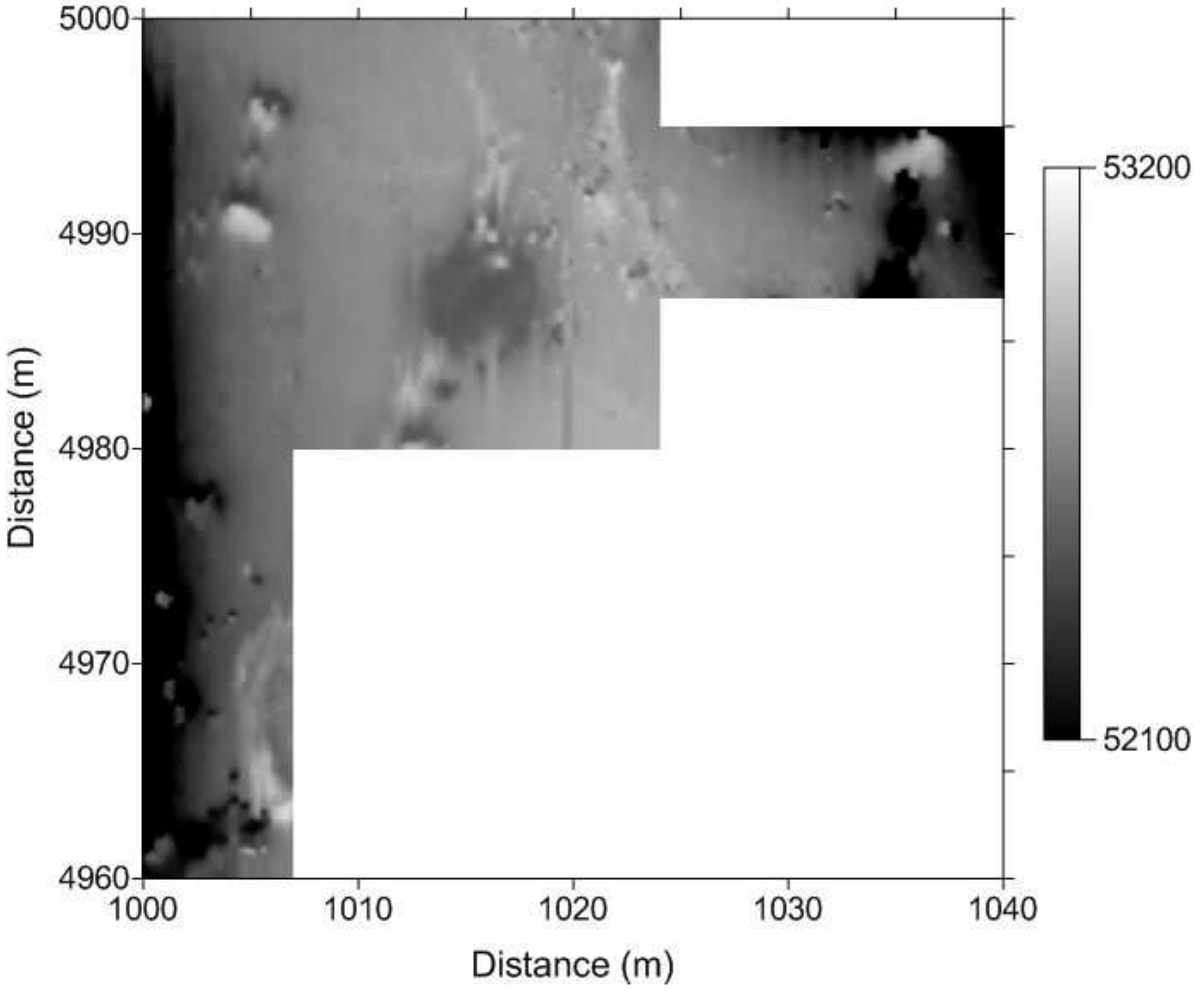


Figure 4.3: The final subsurface survey map of the magnetic data collected at the Seward House Museum National Historic Landmark in Auburn, NY. These data were processed by use of traditional methods in 2 hours time.

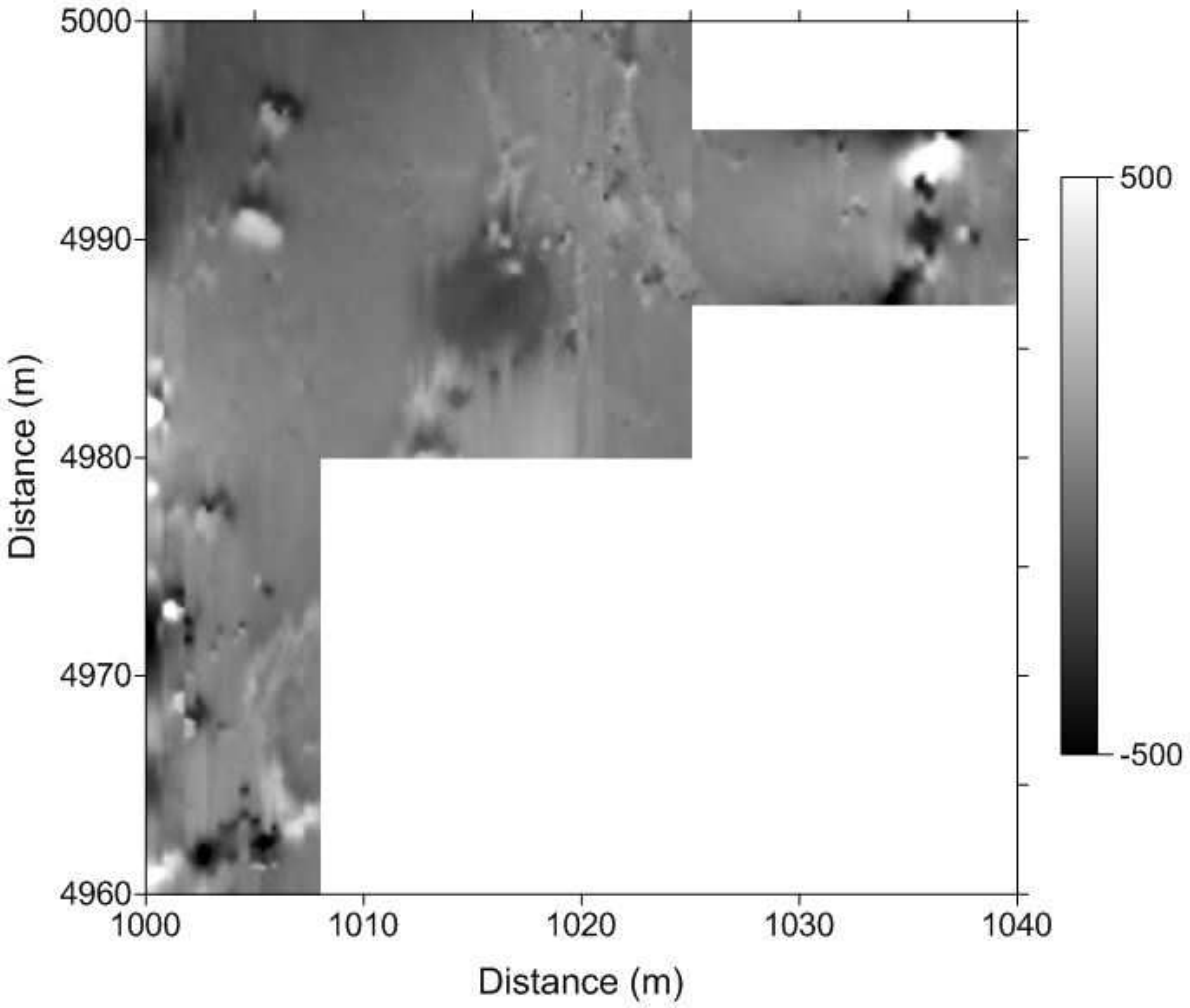


Figure 4.4: The final subsurface survey map of the magnetic data collected at the Seward House Museum National Historic Landmark in Auburn, NY. These data were processed by use of my programs in about 30 minutes.

Chapter 5

Usability Analysis

5.1 Questionnaire Results

I had 4 participants follow the instructions I wrote for processing magnetic data (see Appendix A). I had each participant process two units from a survey and edge match them together in order to determine whether my instructions were clear and if the process was easy to accomplish. I sat with the participants and wrote down notes about any issues that came up and only intervened when they became stuck. I then had each particpant fill out a questionnaire, which was discussed in Chapter 3. The mean of the ratings ($\bar{r}$) for questions 1-3 and their associated standard deviations of the mean ($\bar{\sigma}$) can be seen in Table 5.1. The responses to question 4 are shown in Table 5.2. I will tackle what these results mean and their implications in the following sections.

5.2 Questionnaire Analysis

I will begin with the ratings associated with questions 1-3 from the questionnaire. Question 1 dealt with the task of processing two individual units from a survey. The rating for accomplishing this task was 8.3 ± 0.6, as shown in Table 5.1. This process takes a

Table 5.1: The mean rating and their associated standard deviations of the mean for questions 1-3. Each question asked the user to rate the difficulty of a specific task on a 1-10 scale where 10 corresponded to the task being very easy to accomplish. The ratings help uncover the difficulties of processing magnetic data. The results show that there was a lack of clarity in the instructions and that edge matching units was easier than processing individual units. This difference can be explained by the increased cognitive and kinematic load when processing individual units.

Question	$\bar{r} \pm \bar{\sigma}$
1) Individual processing difficulty	8.3 ± 0.6
2) Edge Matching difficulty	8.5 ± 0.6
3) Clarity of instructions	8.0 ± 0.4

Table 5.2: The comments written by the participants for question 4 of the questionnaire. These responses helped pinpoint the areas where improvements needed to be made in regards to how to go through the data processing steps.

Responses to Question 4
"You should explain how to run a program in Intelij and how to add an image layer and fix the color scale in Surfer."
"Make sure your language is consistent throughout the instructions."
"The instructions need some reworking but the program was pretty easy to use."
"It's pretty clear how to do everything."

longer amount of time compared to edge matching in that there is the use of several pieces of software (MagMap and Surfer). After using one of my programs, the user is confronted with the task of choosing one of several files to finish processing in Surfer. These multiple files are created in case staggering has occured, as discussed in Chapter 3. Once the user has proccessed the data they can view the resulting image and if it is staggered, they will have to choose one of the other files and process it in hopes of achieving a less staggered image. This leads to multiple failed attempts, which increases the cognitive and kinematic load of the participants because they have to keep repeating these final steps to finish processing a unit. Cognitive load refers to the amount of mental activity required to accomplish a task, while kinematic load refers to the degree of physical activity required to accomplish a task (Lidwell *et al.* , 2010). The participant's cognitive load is increased by the amount of reading required (looking for files and reading instructions) and their kinematic load is increased by having to move the mouse, click, and type. There is no way to process each file in Surfer at the same time and there needs to be a range of files with different de-staggering values because the variance of staggering is random and depends on whether the individual who collected the data maintained a constant pace while surveying.

Question 2 addressed the task of edge matching units together and the resulting rating was 8.5 ± 0.6 (see Table 5.1). This task doesn't require the user to use as many programs and follow as many steps compared to the previous task, which reduces the particpant's cognitive and kinematic load. Question 3 dealt with the instructions and, as expected, the rating was not as high as the previous questions. Table 5.1 shows that the rating for the clarity of the instructions was 8.0 ± 0.4. This lower rating was expected because I have never written instructions before and my lack of experience lead to the participants feeling lost and confused at certain points. Since I have experience with processing magnetic data and have gone through the process several times, the instructions were

clear to me. Seeing where the participants struggled allowed me to adjust the language and structure of the instructions to better suit someone who has no experience with this process.

The suggestions left by the participants in question 4 are listed in Table 5.2. Most of the responses targeted the instructions and the lack of clarity within them at certain points. Upon observing the participants I found that they did not have any trouble using my programs, but the lack of experience with processing magnetic data is where confusion arose.

5.3 Eliminating Bias

It is important to address some of the errors associated with this questionnaire in order to comment on the validity in the ratings. Since all of the participants knew me personally, this could have influenced them to give higher ratings than if they did not know me. In order to reduce this bias I told each participant that their answers would be anonymous and that I would rather they provide honest feedback in order to properly address any problems with the instructions, my programs, or anything else that came up during the task. I had the participants fill out the questionnaire alone and waited for them to be finished to prevent any kind of influence on their answers. The sample size of participants is another contributer to error in these ratings. A small sample size was used because there was a time constraint for this project and that it would not have been possible to use a large sample of people. Using a large sample size of people for testing would also be overwhelming in that most of the participants would run into the same confusion that arose for each participant in terms of the errors in the instructions. These ratings and comments were crucial in that having a system that can be understood by someone new defines whether this system will be used in the future.

Chapter 6

Conclusion

The main objective of this project was to ease the task of processing magnetic data collected by Professor Rogers's geophysical archaeology team by creating computer programs that processed the data and eliminated the tedious and time consuming task of copying, pasting, and editing the data manually. I was successfully able to create such programs using Java. Not only did my programs streamline the process of creating a subsurface map, but they also processed the data in such a way that the images produced were more crisp and contained more magnetic signals than those yielded from traditional methods. I was able to create instructions on how to process magnetic data so that someone who had never processed these data could learn how to do it. By having participants follow the instructions and provide feedback on what could be improved I was able to tailor the directions such that they were easy to follow.

These results have significant implications. If the correct software is present on a laptop, processing could be accomplished on site with ease. Any errors that come up can be discovered and rectified before leaving the site. This is advantageous if the site in question is far away and not easily accessed. Another advantage of being able to process data on site is if a subsurface feature that was not expected to be found was discovered,

the team could re-evaluate their surveying plans and focus on this new feature, if desired. Producing sharper images means that these subsurface features will be more distinct and it will be easier to make conclusions on what these features might be.

6.1 Future Work

There are a variety of ways in which my program could be expanded and improved in order to further decrease the workload on the user. This includes adding functions within my program that would accomplish the processing done in Surfer, which is kriging. This method could be added to my program and would save the user from having to finish the processing in Surfer and would only use this piece of software to display the images. This would require a greater understanding on how kriging works and how to implement it correctly within the program.

Another function that could be added to my program involves the method of Upward Continuation. This method involves creating theoretical data by mathematical calculations from the original data. By adding these theoretical data to the original data, it would be possible to discern whether adding these data increases the quality of the subsurface images.

My programs can now be used to streamline data processing in Professor Rogers's geophysical archaeology laboratory. The programs successfully process these data in a time efficient manner and produce clearer subsurface images which will allow for archaeologists to make more insightful deductions on what kinds of features are present.

References Cited

Aspinall, Arnold, Gaffney, Chris F., & Schmidt, Armin. 2008. *Magnetometry for Archaeologists*. Rowman Altamira.

Becker, Helmut, & Fassbinder, Jrg WE. 2015a. In Search for Piramesses-the Lost Capital of Ramesses II. in the Nile Delta (Egypt) by Caesium Magnetometry. *Monuments and Sites*, **6**, 66–70.

Becker, Helmut, & Fassbinder, Jrg WE. 2015b. Magnetometry of a Scythian settlement in Siberia near Cicah in the Baraba Steppe 1999. *ICOMOSHefte des Deutschen Nationalkomitees*, **33**, 168–172.

Brain, Robert M, & Knudsen, Ole. 2007. *Hans Christian Ørsted and the romantic legacy in science: Ideas, disciplines, practices*. Vol. 241. Springer Science & Business Media.

Breiner, Sheldon. 1973. Applications manual for portable magnetometers.

Byksara, A., Arisoy, M, Bekta, , Koak, , & ay, T. 2008. Determination of grave locations in Dedemezari Necropolis (Western Turkey) using magnetic field derivatives. *Archaeological Prospection*, **15**(4), 267–283.

Cammarano, F., Mauriello, P., & Piro, S. 1997. High-resolution geophysical prospecting with integrated methods. The ancient Acropolis of Veio (Rome, Italy). *Archaeological Prospection*, **4**(4), 157–164.

Gaffney, Chris F., Gater, John A., Linford, Paul, Gaffney, Vince L., & White, Roger. 2000. Large-scale systematic fluxgate gradiometry at the Roman city of Wroxeter. *Archaeological Prospection*, **7**(2), 81–99.

Geometrics. 2002. *Magmap 2000 User Guide*. Geometrics Inc., 2190 Fortune Dr. San Jose, California.

Gerard-Little, Peregrine A, Rogers, Michael B, & Jordan, Kurt A. 2012. Understanding the built environment at the Seneca Iroquois White Springs Site using large-scale, multi-instrument archaeogeophysical surveys. *Journal of Archaeological Science*, **39**(7), 2042–2048.

Gilbert, William. 1958. *De magnete*. Courier Corporation.

Griffiths, David. 2012. *Introduction to Electrodynamics*. 4 edn. Addison-Wesley.

Holzner, Steven. 2000. *Java Black Book*. Coriolis Group Books.

Lidwell, William, Holden, Kritina, & Butler, Jill. 2010. *Universal Principles of Design*. Rockport Publishers.

Mahoney, Colleen. 2015. *A Comparative study of Magnetometers in Archaeogeophysics*. B.A. Thesis, Ithaca College.

Mercer, Emily, & Schmidt, Armin. 2001. A magnetometer survey of an Iron Age settlement site at Uppåkra, Skåne, Sweden. *Uppåkrastudier (4). Almqvist & Wiksell International*, 65–78.

Program Creek. 2016. *ArrayList vs. LinkedList vs. Vector*. `http://www.programcreek.com/2013/03/arraylist-vs-linkedlist-vs-vector/`. Accessed 16.02.02.

Rogers, M. B., Baham, J. E., & Dragila, M. I. 2006. Soil iron content effects on the ability of magnetometer surveying to locate buried agricultural drainage pipes. *Applied engineering in agriculture*, **22**(5), 701–704.

Rogers, Michael. 2011. Ch. 8: Archaeological Geophysics: Seeing Deeper with Technology to Compliment Digging. *Archaeology in D*, **3**.

Rogers, Michael B., Faehndrich, Kevin, Roth, Barbara, & Shear, Greg. 2010. Cesium magnetometer surveys at a Pithouse Site near Silver City, New Mexico. *Journal of Archaeological Science*, **37**(5), 1102–1109.

Schmidt, Armin, & Fazeli, Hassan. 2007. Tepe Ghabristan: A Chalcolithic Tell Buried in Alluvium. *Archaeological Prospection*, **14**(1), 38–46.

Sheriff, Steven D., Macdonald, Douglas, & Dick, David. 2010. Decorrugation, edge detection, and modelling of total field magnetic observations from a historic town site, Yellowstone National Park, USA. *Archaeological Prospection*, **17**(1), 49–60.

Software, Golden. 2009. *Surfer Getting Started Guide*. Golden Software, Inc., 809 14^{th} Street, Golden, Colorado.

Stack Overflow. 2016. *When to use LinkedList over ArrayList?* `http://stackoverflow.com/questions/322715/when-to-use-linkedlist-over-arraylist`. Accessed 16.02.02.

Stein, Michael L. 2012. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Science & Business Media.

Stull, Scott, Rogers, Michael, & Batruch, Nik. 2013. Finding Fort Hardy: Combining Documentary Research, Archaeo-Geophysics, and Excavation to Locate a French and Indian War Fort.

Stull, Scott, Rogers, Michael, & Hurley, Kevin. 2014. Colonial Houses and Cultural Identity in New York States Mohawk River Valley. *Archaeological Discovery*, 13–25.

Tabbagh, Jeanne. 2003. Total field magnetic prospection: are vertical gradiometer measurements preferable to single sensor survey? *Archaeological Prospection*, **10**(2), 75–81.

Telford, William Murray, Geldart, L. P., & Sheriff, Robert E. 1990. *Applied Geophysics*. Cambridge University Press.

Woodward, Charles. 2015. *An Archaeogeophysical Study of Washington Hall*. B.A. Thesis, Ithaca College.

Appendix A

Processing Magnetometer Data: Instructions

Important Notes!!

1. Make sure you copy the *.stn files from the RAW folder to the Working folder before you begin processing.
2. Use proper naming convention when naming files.
3. Keep all the files on the network.
4. Ask Professor Rogers for further explanation on the previous items.

A.1 Individual Unit Processing

Step 1: MagMap2000

1. Launch MagMap2000.
2. Click *File* on the menu bar ⇒ Select *Open* ⇒ Choose the appropriate *.stn file for the unit you want processed.
3. Right-click the newly opened window ⇒ Select *Plot Mag Field*.
4. Right-click the newly opened window ⇒ Select *Plot Sensors Setup* ⇒ Select *Sensor 1* **and** *Sensor 2* ⇒ Click *OK*.
5. Click *Filter* on the menu bar ⇒ Select *Remove Drop Outs*.

6. Click *Filter* on the menu bar ⇒ Select Range Despike ⇒ Follow the instructions in the pop-up window and click *OK* ⇒ Click-hold and drag to create the rectangle. **Repeat this step if necessary** (Ask Professor Rogers for assistance if you have questions).

7. Close out of the window and click *File* on the menu bar ⇒ Select *Export* ⇒ Select the following and then click *Export Now*:

   - **Sensor Type**: 2 sensors to cover 2 lines
   - **Sensor Separation**: 0.25
   - **File format**: Surfer
   - **Output format**: X, Y, FIELD, LINE

Step 2: Intelij

** To run a program in Intelij, right click the code of the appropriate program and click 'Run...' (symbol looks like a green play button) **

1. Launch Intelij.

2. Run *Research_Program8*.

3. Find and select the appropriate *.dat file that you exported from the previous section.

4. The program will create one .dat file that has been despiked and destriped and seven other .dat files that have been despiked, destriped, and destaggered by an amount between 1 cm and 7 cm. The number before DESTAG in the file name is the number of centimeters that the unit has been destaggered (e.g. LARD-WORKING-OPTI-U02-DESPIKE-DESTRIPE-3-DESTAG.dat ⇒ has been destaggered by 3 cm).

Step 3: Surfer

1. Launch Surfer

2. Under *Grid* in the menu bar, select *Data...* and select the appropriate file. **Suggestion** Choose the file that has been Despiked and Destriped but has NOT been destaggered because some units might not need destaggering at all.

3. A new window will appear and select the following options:

   (a) X: **Column A: X**

(b) Y: **Column B: Y**

(c) Z: **Column C: T**

(d) Gridding Method: **Kriging**

(e) For the X Direction and Y Direction values: Round to the nearest tens place.

(f) Under Spacing: Use a value of 0.2 for both.

(g) Click OK

4. Under *Map* in the menu bar, click *New* ⇒ select *Image Map...* and select the grid file you created from the previous step.

5. Repeat the items above if the image looks staggered. (Example of what a staggered unit looks like can be seen in Fig. 3.5)

6. If your unit does not look staggered, grid that same *.DAT using the steps above but change the **Spacing** value to 0.05.

****REPEAT STEPS 1-3 UNTIL ALL INDIVIDUAL GRIDS HAVE BEEN PROCESSED BEFORE MOVING ON TO THE NEXT SECTION****

Delete all the other *.DAT files that you did not use for each unit in order to keep your folder clean.

A.2 EdgeMatching

Notes

- Work from the bottom of the survey map to the top (if applicable).
- If you have horizontally adjacent grids, combine them in Surfer before you begin edge matching.

Step 1: Surfer

1. For the top and bottom units you are going to edge match together:
   - In the menu bar: Click *File* ⇒ *Open* ⇒ Select the appropriate *.grd file.
   - A new tab should open up. In the menu bar Click *File* ⇒Select *Save as...* ⇒ Choose the same name as the *.grd file you are using but add "-K" to the file name and save this as a ***.DAT** file.
   - Repeat above for the other unit.

Step 2: Intelij

1. Open Intelij and run the program *Edge_Matching3*.
2. Select the appropriate orientation of the units.
3. Choose the file that corresponds to the bottom unit first, click *Open*, and then choose the file the corresponds to the top. (These are the files you saved in the previous step)
4. After the program finishes, you should have a new *.dat file that ends in "TIP".

Step 3: Back to Surfer

1. Grid the new *.dat file using the information in section **A.1: Step 3: Surfer**.
2. In the menu bar: Click Grid ⇒ Math...
3. Select the top unit *grd file and then select the new "TIP" *.grd file.
4. Make sure the function is 'A + B'
5. Name your output file and click OK.
6. Display the images in Surfer and see if the edge is gone. (Make sure the images are on the same color scale)

Appendix B

Code

B.1 ResearchProgram8

```
import javax.swing.*;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

/**
 * --- Magnetometer Data Processing Program ---
 * created by: Jeffrey Hejna '16
 * email: jeffhejna@gmail.com
 * date: 6/25/2015
 * last updated: 4/23/2016
 *
 * This program will De-stagger units; providing multiple files to test.
 * It will also Destripe and remove edges between horizontally adjacent
 * units by using the Zero Mean Line method to normalize
 * the trends of each transect to 0.
 */


public class Research_Program8 {
    private String fileName;
    private ArrayList<Double> finalXvalues;
    private ArrayList<Double> finalYvalues;
    private ArrayList<Double> finalReadings;
    private ArrayList<Integer> finalLines;
```

```
private Research_Program8(){
    fileName=getFile();
    finalXvalues = new ArrayList<>(60000);
    finalYvalues = new ArrayList<>(60000);
    finalReadings = new ArrayList<>(60000);
    finalLines = new ArrayList<>(60000);
}


/**
 * This method creates a GUI to select the file that
 * the user wants to process.
 * @return The file name
 */
private String getFile(){
    String filename;
    while(true) {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showOpenDialog(null);

        if (result == JFileChooser.APPROVE_OPTION) {
            filename = fileChooser.getSelectedFile().getPath();
            String message = "Is "+ filename + " correct?";
            int reply = JOptionPane.showConfirmDialog(null,message,
            "IMPORTANT!!", JOptionPane.YES_NO_OPTION);
            if(reply == JOptionPane.YES_OPTION){
                break;
            }
        } else if (result == JFileChooser.ERROR_OPTION) {
            JOptionPane.showMessageDialog(null, "An error occurred.");
        } else if(result == JFileChooser.CANCEL_OPTION){
            filename="null";
            break;
        }
    }
    return filename;
}
```

```
/**
 * This method reads the text file, creates arrays for the raw data,
 * removes dropouts, shifts x values by 0.25 cm, creates arrays
 * of data for each sensor, and then calls the
 * Destripe method for both sensors
 * @return 0 if completed; -1 if no file was chosen.
 * @throws Exception
 */
private int readFile() throws Exception {
    if(this.fileName.equals("null")){
        return -1;
    }else{
        Scanner lineScan = new Scanner(new FileInputStream(fileName));
        Scanner fileScan = new Scanner(new FileInputStream(fileName));

        lineScan.nextLine();
        int numLines = 0;
        while (lineScan.hasNextLine()) {
            numLines++;
            lineScan.nextLine();
        }


        double[] X_Raw = new double[numLines];
        double[] Y_Raw = new double[numLines];
        double[] Reading_Raw = new double[numLines];
        int[] Line_Raw = new int[numLines];

        fileScan.nextLine();
        int num=0;
        while(fileScan.hasNextLine()){
            X_Raw[num] = fileScan.nextDouble();
            Y_Raw[num] = fileScan.nextDouble();
            try{
                Reading_Raw[num] = fileScan.nextDouble();
            }catch (Exception e){
                Reading_Raw[num] = 0.0;
                fileScan.next();
                fileScan.next();
            }
            Line_Raw[num] = fileScan.nextInt();

            num++;
```

```
    fileScan.nextLine();
}

ArrayList<Double> Xlist = new ArrayList<>(60000);
ArrayList<Double> Ylist = new ArrayList<>(60000);
ArrayList<Double> Readinglist = new ArrayList<>(60000);
ArrayList<Integer> Linelist = new ArrayList<>(60000);

for (int i = 0; i < X_Raw.length; i++) {
    if (Reading_Raw[i] != 0.0) {
        Xlist.add(X_Raw[i]+0.25);
        Ylist.add(Y_Raw[i]);
        Readinglist.add(Reading_Raw[i]);
        Linelist.add(Line_Raw[i]);
    }
}

double[] Xvalues = new double[Xlist.size()];
double[] Yvalues = new double[Ylist.size()];
double[] Readings = new double[Readinglist.size()];
int[] Line = new int[Linelist.size()];

for (int i = 0; i < Xlist.size(); i++) {
    Xvalues[i] = Xlist.get(i);
    Yvalues[i] = Ylist.get(i);
    Readings[i] = Readinglist.get(i);
    Line[i] = Linelist.get(i);

}

num = 0;
for (int i = 0; i < Line_Raw.length; i++) {
    if (Line[i] < Line[i + 1]) {
        num = i;
        break;
    }
}

double[] leftSensorX = new double[num + 1];
double[] leftSensorY = new double[num + 1];
double[] leftSensorReading = new double[num + 1];
int[] leftSensorLine = new int[num + 1];
```

```
            double[] rightSensorX = new double[Xvalues.length - num - 1];
            double[] rightSensorY = new double[Yvalues.length - num - 1];
            double[] rightSensorReading = new double[Readings.length-num-1];
            int[] rightSensorLine = new int[Line.length - num - 1];


            for (int i = 0; i < num + 1; i++) {
                leftSensorX[i] = Xvalues[i];
                leftSensorY[i] = Yvalues[i];
                leftSensorReading[i] = Readings[i];
                leftSensorLine[i] = Line[i];
            }

            int location = 0;
            for (int i = num + 1; i < Yvalues.length; i++) {
                rightSensorX[location] = Xvalues[i];
                rightSensorY[location] = Yvalues[i];
                rightSensorReading[location] = Readings[i];
                rightSensorLine[location] = Line[i];
                location++;
            }

            Destripe(leftSensorX,leftSensorY,leftSensorReading,
            leftSensorLine);
            Destripe(rightSensorX,rightSensorY,rightSensorReading,
            rightSensorLine);

            return 0;
        }
    }


    /**
     * Takes each line and subtracts the data points from the best fit
     * line in order to normalize it to 0.
     * @param X X values from sensor
     * @param Y Y values from sensor
     * @param Reading Readings from sensor
     * @param Line Line numbers from sensor
     * @throws InterruptedException
     * @throws IOException
     */
```

```
private void Destripe(double[] X, double[] Y,
double[] Reading, int[] Line)
        throws InterruptedException, IOException {

    int lineNum = Line[0];
    int elements, min, max, spot, location;
    while (lineNum >= 0) {

        elements = 0;
        min = 500000000;
        max = -500000000;

        for (int i = 0; i < Y.length; i++) {
            if (Line[i] == lineNum) {
                elements++;
                if (i < min) {
                    min = i;
                }
                if (i > max) {
                    max = i;
                }
            }
        }

        double[] tempX = new double[elements];
        double[] tempY = new double[elements];
        double[] tempReading = new double[elements];
        int[] tempLine = new int[elements];

        spot = 0;
        for (int i = min; i < max + 1; i++) {
            tempX[spot] = X[i];
            tempY[spot] = Y[i];
            tempReading[spot] = Reading[i];
            tempLine[spot] = Line[i];
            spot++;
        }

        double[] regLine = calcRegression(tempY,tempReading);
        location = 0;
        for (int i = 0; i < tempReading.length; i++) {
            tempReading[i] = tempReading[i] - regLine[location];
            location++;
```

```
                finalXvalues.add(tempX[i]);
                finalYvalues.add(tempY[i]);
                finalReadings.add(tempReading[i]);
                finalLines.add(tempLine[i]);
            }
            lineNum--;
        }
    }


    /**
     * This function calculates the best fit line for
     * an individual transect
     * @param Y Y values of the transect
     * @param Reading Magnetic data of a transect
     * @return Array of the values for the best fit line
     */
    private double[] calcRegression(double[] Y, double[] Reading){
        int N = Y.length;
        double sumY=0;
        double sumReading=0;
        double sumYandReading=0;
        double sumMinusYsq=0;

        for (int i = 0; i < Y.length; i++) {
            sumY+=Y[i];
            sumReading+=Reading[i];
        }
        sumY=sumY/N;
        sumReading=sumReading/N;

        for (int i = 0; i < Y.length; i++) {
            sumYandReading+=(Y[i]-sumY)*(Reading[i]-sumReading);
            sumMinusYsq+=(Y[i]-sumY)*(Y[i]-sumY);
        }
        double m = sumYandReading/sumMinusYsq;
        double b = sumReading - m*sumY;

        double[] functionValues = new double[Y.length];
        for (int i = 0; i < Y.length; i++) {
            functionValues[i]= m*Y[i]+b;
        }
```

```
    return functionValues;
}


/**
 * Shifts the transects in the y direction by different values.
 * Values range from 0.1-0.7 cm.
 * odd lines are shifted upward while even lines are shifted downward
 * @throws IOException
 */
private void Destagger() throws IOException{
    double[] Ytemp= new double[finalYvalues.size()];
    for (double i = 1; i < 8; i++) {
        for (int j = 0; j < finalXvalues.size(); j++) {
            Ytemp[j] = finalYvalues.get(j);
        }
        for (int j = 0; j < finalXvalues.size(); j++) {
            if (finalLines.get(j) % 2 != 0) {
                Ytemp[j] = finalYvalues.get(j) + (i / 10);
            } else {
                Ytemp[j] = finalYvalues.get(j) - (i / 10);
            }
        }

        FileWriter fw = new FileWriter(fileName +"-DESPIKE-DESTRIPE-"
        +(int)i+"-DESTAG.dat");
        fw.write("X Y T Line \r\n");
        for (int j = 0; j < finalXvalues.size(); j++) {
            fw.write(finalXvalues.get(j)+" "+ Ytemp[j]+
            " "+finalReadings.get(j) +" "+ finalLines.get(j)+"\r\n");
        }
        fw.close();
    }
}


/**
 * Writes the file of the processed data values
 * @throws IOException
 */
private void writeFile() throws IOException {
    FileWriter fw = new FileWriter(getFileName() +"-DESPIKE-DESTRIPE.dat");
    fw.write("X Y T Line \r\n");
```

```
        for (int k = 0; k < finalReadings.size(); k++) {
            fw.write(finalXvalues.get(k) +" "+ finalYvalues.get(k) +" "
                    + finalReadings.get(k)+" "+ finalLines.get(k)+"\r\n");
        }
        fw.close();
    }


    /**
     * Splits up the filename
     * @return
     */
    private String getFileName(){
        String parts[] =  fileName.split("\\.");
        fileName = parts[0];
        return fileName;
    }


    public static void main(String[] args) throws Exception {
        Research_Program8 processing = new Research_Program8();
        int x = processing.readFile();
        if (x!=-1) {
            processing.writeFile();
            processing.Destagger();
        }
        JOptionPane.showMessageDialog( null, "Done!" );
    }
}
```

B.2 EdgeMatching3

```
import javax.swing.*;
import java.io.*;
import java.util.ArrayList;
import java.util.Objects;
import java.util.Scanner;

/**
 * --- Magnetometer Data Processing Program ---
 * created by: Jeffrey Hejna '16
 * email: jeffhejna@gmail.com
 * date: 6/25/2015
 * last updated: 4/29/2016
 *
 * This program will Edge Match units together. You should be able to
 * only have to use the middle option in the interactive layout
 * since Research_Program8 removes horizontal edges
 * between units. Just add the horizontal units together in Surfer
 * and then just work from bottom to top.
 */


public class Edge_Matching3 {
    private String Topfilename="";
    private String Topfilename2="";
    private String Bottomfilename="";
    private String Bottomfilename2="";

    private ArrayList<Double> XvaluesTop = new ArrayList<>(60000);
    private ArrayList<Double> YvaluesTop= new ArrayList<>(60000);
    private ArrayList<Double> ReadingsTop = new ArrayList<>(60000);

    private ArrayList<Double> XvaluesBottom = new ArrayList<>(60000);
    private ArrayList<Double> YvaluesBottom= new ArrayList<>(60000);
    private ArrayList<Double> ReadingsBottom = new ArrayList<>(60000);

    private ArrayList<Double> finalXvalues = new ArrayList<>(60000);
    private ArrayList<Double> finalYvalues = new ArrayList<>(60000);
    private ArrayList<Double> finalReadings = new ArrayList<>(60000);

    private double minX;
    private double maxX;
```

```
    private double minY;
    private double maxY;

    private double leftCornerReading;
    private double rightCornerReading;

    private double mean;

    private Edge_Matching3(int choice) throws FileNotFoundException {
        if(choice==1||choice==2||choice==3) {
            Bottomfilename = getFileName(1);
            Topfilename = getFileName(4);

        }else if(choice==4){
            Bottomfilename = getFileName(2);
            Bottomfilename2 = getFileName(3);
            Topfilename = getFileName(4);
        }else{
            Bottomfilename = getFileName(1);
            Topfilename = getFileName(5);
            Topfilename2 = getFileName(6);
        }
    }


    /**
     * Creates an interactive layout in which the user
     * can pick which orientation should be used.
     * ** The middle option should now be the only one you have to use!**
     * @return
     */
    private static int getLayout(){
        StdDraw.setXscale(-200,200);
        StdDraw.setYscale(-200,200);

        //middle choice
        StdDraw.filledRectangle(0,10,20,10);
        StdDraw.filledRectangle(0,-11,20,10);

        //bottom left
        StdDraw.filledRectangle(-160,-160,20,10);
        StdDraw.filledRectangle(-119,-160,20,10);
        StdDraw.filledRectangle(-140,-139,20,10);
```

```
        StdDraw.filledRectangle(140,-160,20,10);
        StdDraw.filledRectangle(160,-139,20,10);
        StdDraw.filledRectangle(119,-139,20,10);

        //top left choice
        StdDraw.filledRectangle(-140,160,20,10);
        StdDraw.filledRectangle(-160,139,20,10);

        //top right choice
        StdDraw.filledRectangle(140,160,20,10);
        StdDraw.filledRectangle(160,139,20,10);

        StdDraw.text(0,190,"Click which arrangement the grids
         you are edge matching are in");
        int choice=0;
        double X;
        double Y;
        while (true) {
            if (StdDraw.mousePressed()) {
                X = StdDraw.mouseX();
                Y = StdDraw.mouseY();
                break;
            }
        }
        if(X>=-50 && X<=50 && Y<=50 && Y>=-50){
            choice=1;
        }else if(X>=-200 && X<=-100 && Y<=200 && Y>=100){
            choice=2;
        }else if(X>=100 && X<=200 && Y<=200 && Y>=100){
            choice=3;
        }else if(X>=-200 && X<=-100 && Y<=-100 && Y>=-200){
            choice=4;
        }else if(X>=100 && X<=200 && Y>=-200 && Y<=-100){
            choice=5;
        }
        return choice;
    }
```

```
/**
 * This function creates GUIs in which the user will select
 * the units they want edge matched,
 * depending on the choice they made in getLayout().
 * @param num
 * @return -1 if both files aren't chosen.
 */
private String getFileName(int num){
    String filename;
    while(true) {
        JFileChooser fileChooser = new JFileChooser();
        int result = fileChooser.showOpenDialog(null);

        if (result == JFileChooser.APPROVE_OPTION && num==1) {
            fileChooser.setDialogTitle("Bottom Unit Choice");
            filename = fileChooser.getSelectedFile().getPath();

            String message = "Is "+ filename +
            " correct for the BOTTOM unit?";

            int reply = JOptionPane.showConfirmDialog(null,message,
            "IMPORTANT!!",JOptionPane.YES_NO_OPTION);

            if(reply == JOptionPane.YES_OPTION){
                break;
            }

        }else if(result==JFileChooser.APPROVE_OPTION && num==2) {
            fileChooser.setDialogTitle("Left Bottom Unit Choice");
            filename = fileChooser.getSelectedFile().getPath();

            String message = "Is "+ filename +
            " correct for the LEFT BOTTOM unit? (LEFT one)";

            int reply = JOptionPane.showConfirmDialog(null,message,
            "IMPORTANT!!",JOptionPane.YES_NO_OPTION);

            if(reply == JOptionPane.YES_OPTION){
                break;
            }
        }else if(result==JFileChooser.APPROVE_OPTION && num==3) {
            fileChooser.setDialogTitle("Right Unit Choice");
            filename = fileChooser.getSelectedFile().getPath();
```

```
            String message = "Is " + filename +
            " correct for the RIGHT BOTTOM unit";

            int reply = JOptionPane.showConfirmDialog(null, message,
            "IMPORTANT!!", JOptionPane.YES_NO_OPTION);

            if (reply == JOptionPane.YES_OPTION) {
                break;
            }
        }else if(result==JFileChooser.APPROVE_OPTION && num==4) {
            filename = fileChooser.getSelectedFile().getPath();
            fileChooser.setDialogTitle("Top Unit Choice");

            String message = "Is "+ filename +
            " correct for the TOP unit";

            int reply = JOptionPane.showConfirmDialog(null,message,
            "IMPORTANT!!",JOptionPane.YES_NO_OPTION);

            if(reply == JOptionPane.YES_OPTION){
                break;
            }
        }else if(result==JFileChooser.APPROVE_OPTION && num==5) {
            fileChooser.setDialogTitle("Left Top Unit Choice");

            filename = fileChooser.getSelectedFile().getPath();
            String message = "Is "+ filename +
            " correct for the LEFT TOP unit";

            int reply = JOptionPane.showConfirmDialog(null,message,
            "IMPORTANT!!",JOptionPane.YES_NO_OPTION);

            if(reply == JOptionPane.YES_OPTION){
                break;
            }
        }else if(result==JFileChooser.APPROVE_OPTION && num==6) {
            fileChooser.setDialogTitle("Right Top Unit Choice");
            filename = fileChooser.getSelectedFile().getPath();
            String message = "Is "+ filename +
             " correct for the RIGHT TOP unit";

            int reply = JOptionPane.showConfirmDialog(null,message,
            "IMPORTANT!!",JOptionPane.YES_NO_OPTION);
```

```
            if(reply == JOptionPane.YES_OPTION){
                break;
            }
        }else if (result == JFileChooser.ERROR_OPTION) {
            JOptionPane.showMessageDialog(null, "An error occurred.");
        }else if(result == JFileChooser.CANCEL_OPTION){
            filename="null";
            break;
        }
    }
    return filename;
}


/**
 * This function gets the values for the corners of the units
 * @param option
 * @throws FileNotFoundException
 */
private void getCorners(int option) throws FileNotFoundException {
    Scanner scanner = new Scanner(new FileInputStream(Topfilename));
    ArrayList<Double> X = new ArrayList<>();
    ArrayList<Double> Y = new ArrayList<>();
    ArrayList<Double> Reading = new ArrayList<>();

    while (scanner.hasNextLine()){
        X.add(scanner.nextDouble());
        Y.add(scanner.nextDouble());
        Reading.add(scanner.nextDouble());
        scanner.nextLine();
    }
    scanner.close();
    if(option==5){
        Scanner scanner2 = new Scanner(
        (new FileInputStream(Topfilename2)));

        scanner2.nextLine();
        while(scanner2.hasNextLine()){
            X.add(scanner2.nextDouble());
            Y.add(scanner2.nextDouble());
            Reading.add(scanner2.nextDouble());
            scanner2.nextLine();
```

```
            }
            scanner2.close();
        }

        minY=Y.get(0);
        maxY=Y.get(Y.size()-1);
        minX = XvaluesTop.get(0);
        maxX = XvaluesTop.get(XvaluesTop.size()-1);

        for (int i = 0; i < X.size(); i++) {
            if(Y.get(i)==maxY && X.get(i)==minX){
                leftCornerReading = Reading.get(i);
            }else if(Y.get(i)==maxY && X.get(i)==maxX){
                rightCornerReading = Reading.get(i);
            }
        }

        for (int i = 0; i < ReadingsTop.size(); i++) {
            mean += ReadingsTop.get(i);
        }
        mean = mean/ReadingsTop.size();
    }


    /**
     * Grabs the information for the units depending on what choice is
     * made from the interactive layout.
     * @param choice Choice from the layout
     * @throws FileNotFoundException
     */
    private void getGridInfo(int choice) throws FileNotFoundException {
        Scanner Topfile = new Scanner(new FileInputStream(Topfilename));
        while (Topfile.hasNextLine()){
            XvaluesTop.add(Topfile.nextDouble());
            YvaluesTop.add(Topfile.nextDouble());
            ReadingsTop.add(Topfile.nextDouble());
            if (YvaluesTop.get(YvaluesTop.size()-1)>YvaluesTop.get(0)){
                XvaluesTop.remove(XvaluesTop.size() - 1);
                YvaluesTop.remove(YvaluesTop.size() - 1);
                ReadingsTop.remove(ReadingsTop.size() - 1);
                break;
            }
            Topfile.nextLine();
```

```
        }
        Topfile.close();

        if(choice==5){
            Scanner Topfile2 = new Scanner(
            new FileInputStream(Topfilename2));

            while(Topfile2.hasNextLine()){
                XvaluesTop.add(Topfile2.nextDouble());
                YvaluesTop.add(Topfile2.nextDouble());
                ReadingsTop.add(Topfile2.nextDouble());
                if (YvaluesTop.get(YvaluesTop.size()-1)>YvaluesTop.get(0)){
                    XvaluesTop.remove(XvaluesTop.size() - 1);
                    YvaluesTop.remove(YvaluesTop.size() - 1);
                    ReadingsTop.remove(ReadingsTop.size() - 1);
                    break;
                }
                Topfile2.nextLine();
            }
        }

        getCorners(choice);

        ArrayList<Double> tempXvalues = new ArrayList<>(60000);
        ArrayList<Double> tempYvalues = new ArrayList<>(60000);
        ArrayList<Double> tempReadings = new ArrayList<>(60000);
        if(choice!=4){
            Scanner Bottomfile = new Scanner(
            new FileInputStream(Bottomfilename));

            while (Bottomfile.hasNextLine()){
                tempXvalues.add(Bottomfile.nextDouble());
                tempYvalues.add(Bottomfile.nextDouble());
                tempReadings.add(Bottomfile.nextDouble());
                Bottomfile.nextLine();
            }
            Bottomfile.close();

        }else{
            Scanner Bottomfile1 = new Scanner(
            new FileInputStream(Bottomfilename));

            while (Bottomfile1.hasNextLine()){
```

```
            tempXvalues.add(Bottomfile1.nextDouble());
            tempYvalues.add(Bottomfile1.nextDouble());
            tempReadings.add(Bottomfile1.nextDouble());
            Bottomfile1.nextLine();
        }
        Bottomfile1.close();

        Scanner Bottomfile2 = new Scanner(
        new FileInputStream(Bottomfilename2));

        while (Bottomfile2.hasNextLine()){
            tempXvalues.add(Bottomfile2.nextDouble());
            tempYvalues.add(Bottomfile2.nextDouble());
            tempReadings.add(Bottomfile2.nextDouble());
            Bottomfile2.nextLine();
        }
        Bottomfile2.close();
    }

    for (int i = 0; i < tempReadings.size(); i++) {
        if(Objects.equals(tempYvalues.get(i),minY)){
            XvaluesBottom.add(tempXvalues.get(i));
            YvaluesBottom.add(tempYvalues.get(i));
            ReadingsBottom.add(tempReadings.get(i));
        }
    }
}


/**
 * Creates the data that will be put into the -TIP file
 */
private void EdgeMatch(){
    finalXvalues.add(minX);
    finalYvalues.add(maxY);
    finalReadings.add(mean - leftCornerReading);

    finalXvalues.add(maxX);
    finalYvalues.add(maxY);
    finalReadings.add(mean - rightCornerReading);

    for (int i = 0; i < XvaluesTop.size(); i++) {
        for (int j = 0; j < XvaluesBottom.size(); j++) {
```

```
                if(Objects.equals(XvaluesTop.get(i), XvaluesBottom.get(j)) &&
                        (XvaluesTop.get(i)-XvaluesTop.get(i).intValue()==0 ||
                                XvaluesTop.get(i)-
                                XvaluesTop.get(i).intValue()==0.25 ||

                                XvaluesTop.get(i)-
                                XvaluesTop.get(i).intValue()==0.5 ||

                                XvaluesTop.get(i)-
                                XvaluesTop.get(i).intValue()==0.75)){
                    finalXvalues.add(XvaluesTop.get(i));
                    finalYvalues.add(YvaluesTop.get(i));
                    finalReadings.add(ReadingsBottom.get(j)
                    -ReadingsTop.get(i));
                }
            }
        }
    }


    /**
     * Creates a -TIP file that will be used to finish edge matching
     * in Surfer
     * @throws IOException
     */
    private void writeFile() throws IOException {
        FileWriter fw = new FileWriter(getFileName1() +"-TIP.dat");
        fw.write("X Y T Line \r\n");
        for (int k = 0; k < finalReadings.size(); k++) {
            fw.write(finalXvalues.get(k) +" "+ finalYvalues.get(k) +
            " "+ finalReadings.get(k)+"\r\n");
        }
        fw.close();
    }


    /**
     * Splits the file name so it can be used in making the new TIP file
     * @return
     */
    private String getFileName1(){
        String parts[] =  Topfilename.split("\\.");
        Topfilename = parts[0];
```

```
        return Topfilename;
    }


    public static void main(String[] args) throws IOException {
        int choice = getLayout();
        Edge_Matching3 edgeMatching = new Edge_Matching3(choice);
        edgeMatching.getGridInfo(choice);
        edgeMatching.EdgeMatch();
        edgeMatching.writeFile();
        JOptionPane.showMessageDialog(null,"You can now exit the windows");

    }
}
```