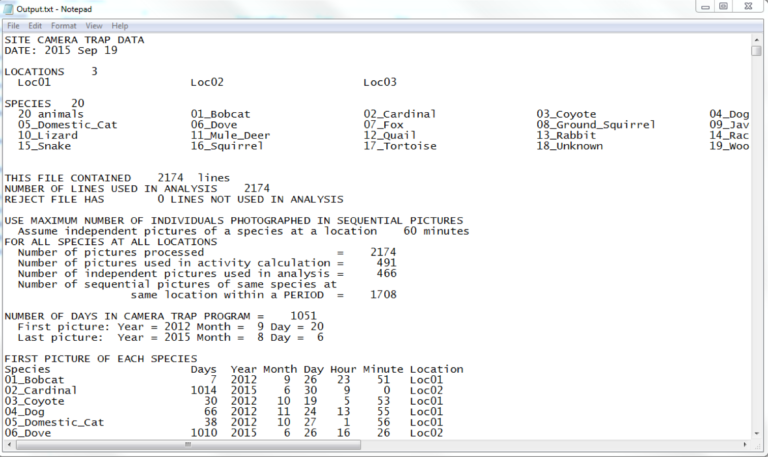
SANIMAL – Scientific Animal Image Analysis

David Slovikosky

**Background:**

In May 2016 I was made aware that the School of Natural Resources and the Environment (*SNRE*) department at the University of Arizona was using outdated software to analyze images taken by camera traps. These motion detector-equipped cameras are set up in the wild and are left to take pictures for many months. After being retrieved, the images are collected and sorted by various researchers from high school students, to university professors, to retired people who do it as a hobby. Images are sorted or *tagged* by hand with GPS location, species, and number of animals. These researchers then aggregate their data into collections which are manually stored on hard drives or servers. Finally, analysis software is run on the tagged data to produce meaningful output used for research.

Originally this task was accomplished with a suite of FORTRAN programs written by Dr. Jim Sanderson. The heart of this software suite is *DataAnalyze*, a program that takes sorted data as input and creates a large text file as output. This *output.txt* file contains thousands of lines of summary statistics on the data in the form of ASCII charts which is heavily used by the SNRE department. A sample screenshot of this output is shown below:



This software has been used for many years and is currently kept up-to-date by Dr. Sanderson. Updates are released as researchers find bugs in the existing calculations or define a new analysis they would like to see. Unfortunately, this software leaves much to be desired. More specifically, *DataAnalyze* is lacking in the following areas:

1. The source code is in FORTRAN and not easily available for custom modification
2. The sorting process for data is very tedious
3. Command line execution of the program can easily crash if the inputs are incorrectly formatted without any error description of what went wrong
4. Output is in text format only and can be challenging to parse
5. Analysis can only be performed on local data making it difficult to collaborate

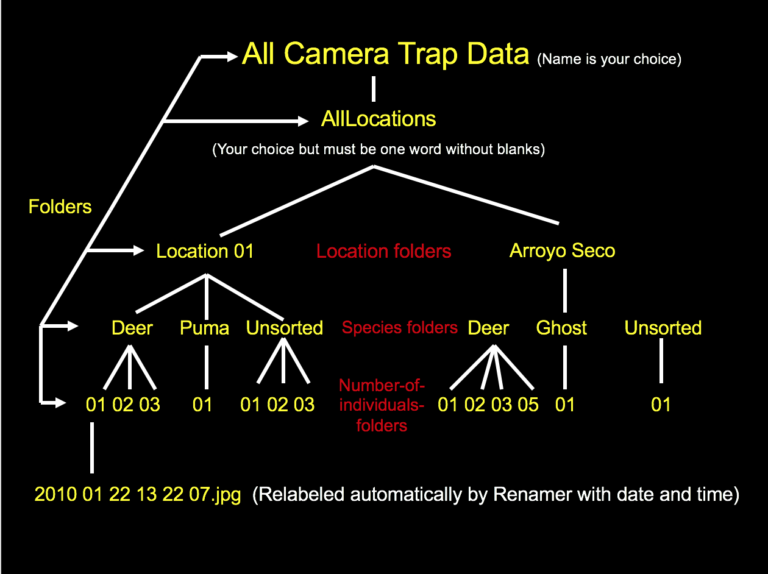
***DataAnalyze* Deficiencies:**

*(1) DataAnalyze* is written in FORTRAN and the source code is currently private. While FORTRAN was a cutting-edge language at its time, it has aged and does not provide many features standard in modern languages. Most importantly, it is challenging to create a user interface in FORTRAN which is essential in modern applications. Because the software is closed source, any modification to *DataAnalyze* is impossible. This means there is no way to see how any of the analysis are done besides the documentation given by Dr. Sanderson. While this documentation does give some insight on how the numbers are computed, it can be difficult to reverse engineer many of the numbers.

(2) As previously stated, images taken on camera traps must be *tagged* with location, species, and animal count. Locations are comprised of a name and UTM coordinates. Species are given by a name, and finally count specifies the number of the species in the image. The image date and time taken must also be extracted from the image metadata. *DataAnalyze* performs this *tagging* process by parsing a directory structure as well requiring an external tool to extract image date and time taken. The *tagging* process begins by renaming all images in the following format: “Year Month Day Hour Minute Second.jpg” by a third-party program named “Renamer”. This software simply reads the image file metadata and renames the file accordingly. All credit for this program is found below:

https://www.den4b.com/products/renamer

Renamed images are then copied into a directory structure pictured below (taken from Dr. Sanderson’s HowTo slides):



This directory structure is then parsed by *DataAnalyze*. Location UTM coordinates are read in from a separate file which is provided as a standard-in argument. These coordinates must be in a very specific format to be parsed correctly, otherwise the FORTRAN software will crash. After a few seconds of execution, *DataAnalyze* creates a few output text files, of which *output.txt* contains most of the analysis. This entire process of renaming the images and then copying them into a multi-layer directory structure is incredibly tedious and lacks scalability. Images that contain multiple species are duplicated in the directory structure leading to data redundancy.

(3) *DataAnalyze* is a Windows executable which is run from the command line. This immediately poses a problem to anyone using a Mac or Unix based system, because they simply cannot use the software. Users must open a terminal and answer questions asked by the software such as, “Enter folder name containing all camera trap locations, then ENTER”. There are many challenges when using command line applications especially to users who lack technical knowledge, and many of the users using *DataAnalyze* have never used a command line-based program. These challenges include understanding relative vs absolute file paths, remembering capitalization matters, and being able to interpret error messages. *DataAnalyze* does not provide any detection for user error and can often crash as a result with a cryptic FORTRAN error message. All these specifics mean the average user cannot easily learn how to use the software and needs to be taught by someone who has used the software.

(4) The output created by *DataAnalyze* is simple ASCII. This output is incredibly difficult to parse, both by humans and computers. Many researchers want to do further analysis on the output created by *DataAnalyze*, but due to the irregular nature of the text writing a script to read all the values can be very time consuming. Output lines are printed with many arbitrary restrictions, such as location and species names are capped around 26 characters and padded with spaces if too short. This string length value varies from chart to chart. Some charts are very wide due to having many columns and *DataAnalyze* splits these charts in two with half the data in the first, and half in the second. After talking with some researchers who use this software, one said “We just copy the ASCII charts straight into Microsoft excel and let excel’s built in parser parse the chart’s values into cells. We then graph the data using excel.” In one small study, 2,500 images were analyzed which resulted in an *output.txt* file 3000 lines long. Many of the charts featured in the output contained the same data but aggregated in many different possible permutations. As an example, some charts group images taken by month, and others group images by season. This results in two separate ASCII charts which are scattered somewhere in the 3000 lines of output. In larger studies, millions of images are often involved which leads to *output.txt* files that are incredibly large. The output of *DataAnalyze* is static and non-interactive. This makes the analysis software very difficult to use for anyone untrained. The SNRE department has a special class each year that is designed to teach citizen scientists how to understand and parse the *output.txt* file which should never be the case with analysis software.

(5) Researchers rarely work alone. There are several different projects currently going on in the SNRE department, each with a unique set of researchers, cameras, and species to track. These projects want to store all their images in a single location where analysis can be done over data from all researchers. Because *DataAnalyze* only runs on local data, one massive, terabyte sized directory structure is stored on a server. Citizen scientists first sort their portion of the data and upload it to the server. The folders are merged, and the image collection is updated. Later a researcher can execute the software on the directory. This system is flawed for many reasons. Firstly, researchers who may live around the world need to “upload” their images to the server by sending USB drives in the mail to someone who works near to the server or upload their data to a Dropbox style service. Secondly, this setup completely disregards data security. Many projects need their images to be kept private, because hunters could abuse knowledge of species movements to track specific species.

**Scientific Animal Image Analysis (*SANIMAL*) Swing:**

To solve these deficiencies, it was clear that *DataAnalyze* needed to be fully re-engineered and re-written from scratch. To begin engineering *SANIMAL*, a more modern language needed to be chosen. Three major languages were considered, C/C++, C#, and Java. C/C++ has the advantage of being efficient and supports a wide variety of libraries to create user interfaces. Unfortunately, C/C++ has compatibility problems when being deployed to a wide variety of operating systems such as Unix, Mac, and Windows. C# has the best user interface creation tools when being designed in Visual Studio. The language also supports modern language features such as data bindings and lambda functions. Unfortunately, C# is only compatible with Windows because it runs on the .NET framework and therefore was not chosen either. Finally, Java was considered for use in *SANIMAL*. Java supports a wide variety of operating systems including Unix, Mac, and Windows while also having a wide variety of user interface frameworks such as the Advanced Windowing Toolkit (AWT), Swing, and JavaFX. One personal reason Java was considered is because I had taken a class, Compute Science 335: Object Oriented Development and Design the year before. This course teaches proper objected-oriented design principles as well as user interface design using Java’s Swing framework. Because of this experience, it was decided to use Java instead of C/C++.

The first version of *SANIMAL* addressed many, but not all problems inherent in *DataAnalyze*. More specifically, it had the following advantages:

1. *SANIMAL* was written in Java using version control
2. *SANIMAL* used a user interface making the data sorting process much easier
3. New output was introduced to increase analysis productivity

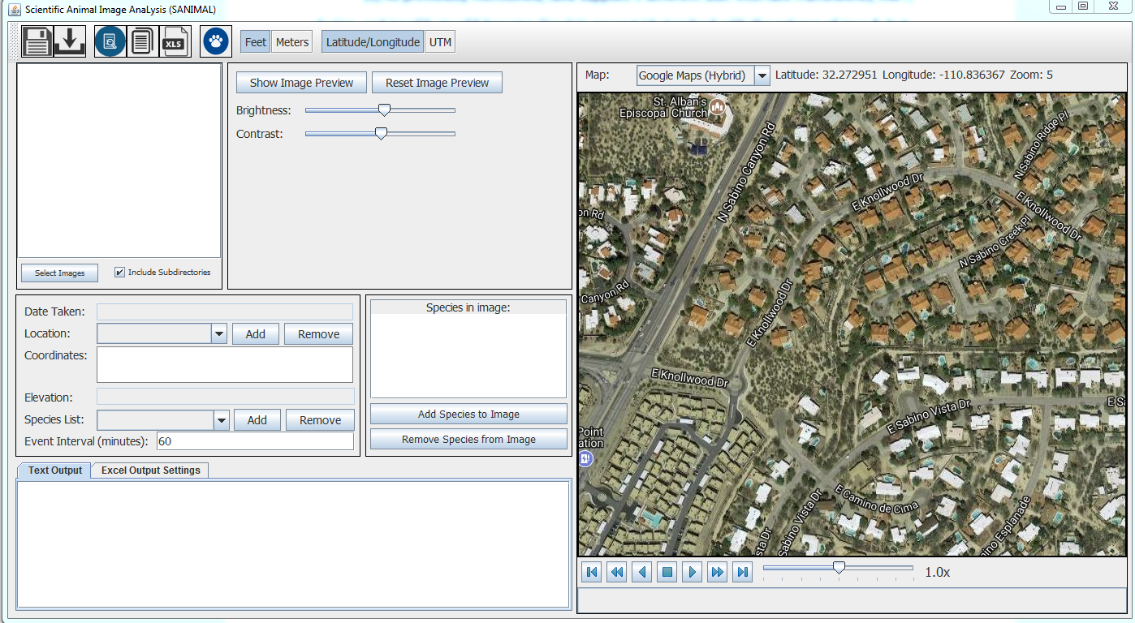
***SANIMAL* Swing in Detail:**

(1) Since Java was chosen, a user interface framework had to be selected. As previously mentioned, Java supplies 3 different UI frameworks, AWT, Swing, and JavaFX. JavaFX became Oracle’s newest UI standard with the release of Java 8, but this project began before Java 8 was popular. AWT was Java’s original graphics library but no longer widely used and heavily deprecated. Due to my experience with Swing, Swing was chosen for the first version of *SANIMAL*. Because Java is supported on most operating systems, Swing would allow *SANIMAL* to use one unified interface for almost all operating systems.

*SANIMAL* was engineered with object-oriented design principles in mind. Most notably, the Observer-Observable, Model-View-Controller, and Singleton design patterns were heavily used. This made the code very robust to new changes and new features were easily implemented. All code was documented using JavaDoc increasing readability further. It was quickly made evident that this project would take much longer to develop than the original single semester allotted. Writing and documenting the code clearly was an essential component in ensuring software longevity.

It was decided to use GitHub to perform version control on the code which was not present in the original *DataAnalyze* software. GitHub allows developers to collaborate on the development of a project as well as providing a detailed list of changes in every code commit. Using GitHub also allows the project to be open source, which enables any user to download and modify the code to their liking.

(2) Instead of using a directory structure, a user interface was created to tag data. A screenshot of the UI is pictured below:



The UI allows users to tag data using the following steps:

1. New locations are added to the program for each location data was collected at. A location consists of a name, a position, and an elevation.
2. A new species is added for each species photographed in the data. A species only consists of a common or scientific name.
3. Images are imported using the select image button.
4. The user goes through each image in the list view and selects the location the image was taken at. Species are added by selecting a species from the list and selecting “Add Species to Image” which prompts users to specify how many of a given species are in the image. The date and time taken are automatically read from the image EXIF metadata. This removes the requirement to use ReNamer to give images a specific name based on date and time.

One major problem encountered when tagging camera trap images is that night time photos may be very dark on the edges, making it difficult to see what species triggered the motion detector. If a user wants to tag a dark image using *DataAnalyze*, an external program such as photoshop must be used to temporarily adjust brightness, contrast, and zoom of the image. To address this, *SANIMAL* can open a preview window of the image which supports temporary zoom, brightness, and contrast adjustments. This makes the tagging process much more efficient because users no longer need external tools to view dark images.

Another problem encountered is that users need a way to save their tagging progress to continue later without needing to keep the program open. *DataAnalyze* did not have this problem, because users tagged images using a directory structure which is automatically stored to disk. If a user wants to save the current program state for later, *SANIMAL* has a save button which serializes its internal data model and saves it into a file. This file can then be opened through the program and all data such as species, locations, and images are loaded in.

It quickly became clear that researchers from around the world were going to use this software, and therefore it was important to support different units of measurement. Elevation can be displayed in feet or meters and location can be displayed in latitude/longitude or UTM. When importing photos, *SANIMAL* asks the user whether they are entering location position in UTM or latitude/longitude, and if elevation is given in feet or meters. This small change made the software more usable than *DataAnalyze*, which only supported UTM and meters.

(3) The largest flaw inherent in *DataAnalyze* is its inability to clearly show analysis results. At the beginning of the project, analysis requirements were to simply recreate the output *DataAnalyze* already created. Recreating the output allowed *SANIMAL* to appeal to researchers used to the old system, and then build upon it. Because source code to *DataAnalyze* was not available, the *output.txt* was reverse engineered using Dr. Sanderson’sdocumentation as well as trial and error. In the reverse engineering process, many major bugs were discovered in *DataAnalyze* unknown to researchers at the time. *SANIMAL*’s output recreation fixed all bugs encountered.

To improve upon analysis, a map was included which displays a marker at each location on the map. The map allows selection from a list of various map providers like google maps, open street map, and virtual earth. Each provider provides a different set of map tiles which can be used to view street, satellite, or terrain data. This map allows playback of image data based on date and time taken. A progress bar lets users to watch the data like a video from the first image’s date to the last one. When the playback gets close an image’s date taken, an image thumbnail is displayed at the location the image was taken. This allows researchers to visualize where images were taken and what time throughout the study.

***SANIMAL* Swing Deficiencies:**

After some basic testing was done on the first version of *SANIMAL*, it was clear that many issues were not fully addressed, and many more additional features were requested. Most importantly, *SANIMAL* did not use a database backend which made it difficult to collaborate. Species and location lists needed to be created or loaded in manually every time the software was opened. Images were not directly tagged with EXIF metadata, so any species and location data were purely stored inside *SANIMAL* or a saved project file. This meant it was impossible to share images with other researchers without passing around project files. While the image tagging process was much faster than on the original *DataAnalyze* tool, the process was still tedious. Many mouse clicks were required to create a species and tag it onto an image. In the majority of cases, each image contained one single species, however there was no quick way of indicating this. The analysis provided by *SANIMAL* was still very basic and, other than the map, the same as *DataAnalyze*. It was clear that the first iteration of *SANIMAL* was insufficient, and a second iteration of the software was needed.

***SANIMAL FX***: