

User Interface Design Patterns for Wildlife-Related Camera Trap Image Analysis

Saul Greenberg

Department of Computer Science
University of Calgary
2500 University Dr. NW
Calgary, AB, Canada, T2N 1N4
saul@ucalgary.ca

Theresa Godin

Freshwater Fisheries Society of BC
Research Evaluation & Development
Section, Rm 315-2202 Main Mall,
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
Theresa.Godin@gofishbc.com

Jesse Whittington

Parks Canada
Banff National Park
Box 900, Banff, Alberta
T1L 1K2 Canada
jesse.whittington@canada.ca

ABSTRACT

This paper describes and explains *design patterns* for software that supports how analysts can efficiently inspect and classify camera trap images for wildlife-related ecological attributes. Broadly speaking, a design pattern identifies a commonly occurring problem, and a general reusable design approach to solve that problem. A developer can then use that design approach to create a specific software solution appropriate to the particular situation under consideration. In particular, design patterns for camera trap image analysis by wildlife biologists address solutions to commonly occurring problems they face while inspecting a large number of images and entering ecological data describing image attributes. We developed design patterns for image classification based on our understanding of biologists' needs that we acquired over eight years during development and application of the freely-available Timelapse image analysis system. For each design pattern presented, we describe the problem, a design approach that solves that problem, and a concrete example of how Timelapse addresses the design pattern. These design patterns can inform the design of other camera trap systems, and can help biologists assess how competing software products address their project-specific needs along with determining an efficient workflow.

KEYWORDS

Camera traps, image inspection, data encoding and acquisition, tagging, wildlife monitoring, design patterns, human computer interaction, experience design.

1. INTRODUCTION

Camera traps – also known as wildlife, remote, field, or trail cameras – are increasingly used to address a broad range of ecological research and field monitoring applications (Steenweg et al., 2017). Their basic idea is deceptively simple. First, cameras are located at strategic stations within a study site, where they are positioned to capture activities occurring within a particular field of view. Second, cameras are set up to take images automatically in one of two ways: a *timelapse* mode where images are taken at regular intervals, or a *motion triggering* mode where one or more images are taken whenever movement is detected in the scene. Third, cameras are serviced after a period of time (weeks or months), where field personnel change camera batteries and retrieve the image-laden SD cards. Fourth, analysts review the set of retrieved images. The analyst examines each image for attributes of interest, and encodes those as descriptive or quantitative data. This step is also known as *tagging*. Finally, that data – usually managed and stored in a database or spreadsheet – becomes the input for the data processing (including statistical analysis) particular to the project.

This paper is primarily concerned with the fourth tagging step described above: how analysts examine images and encode its attributes of interest as data. The number of images typically collected is voluminous: thousands or tens of thousands of images per retrieved camera card easily accumulate to hundreds of thousands of images per project). Consequently, image examination and data encoding is laborious, time-intensive, error-prone, and expensive. It is no wonder that international survey respondents identified image classification as the top challenge in camera trapping (Glover-Kapfer, et al., 2019).

Recent research seeks to remedy this burden via automated image recognition, where promising wildlife species detection and identification rates have been reported (e.g., Norouzzadeha, et al., 2018; Schneider et al, 2018, Tabak et al., 2018; Yousif et al, 2019). Unfortunately, image recognition for camera traps is still in its formative stage. It is limited in what can be recognized. For example, somewhat easy to extremely difficult recognition challenges range from: detecting if wildlife is present, identifying the species, identifying individuals, determining animal health, to distinguishing animal behaviors. Image recognition also requires training on domain-specific images, incurs varying rates of classification errors (false positives and false negatives), performs less well with new camera placements (due to different backgrounds), and is currently poorly integrated in the analyst's software and workflow. Even if it were available, analysts would still have to verify recognition predictions and correct erroneous ones. Manual methods will likely dominate for years to come, especially when additional attributes beyond simple species detection are required (but see Section 8 below).

Cite as:

Greenberg, S., Godin, T. and Whittington, J. (2019) **User Interface Design Patterns for Wildlife-Related Camera Trap Image Analysis**. Report 2019-1114-06, Department of Computer Science, University of Calgary, Calgary, AB Canada. Available via <http://hdl.handle.net/1880/110687> or DOI: <http://dx.doi.org/10.11575/PRISM/36783> . July 29.

In the past, analysts resorted to off-the-shelf generic software when tagging images, such as a stock image viewer to view images, and a separate spreadsheet package to record data. More recently, researchers and corporations have developed specialized software packages to support camera trap analysis (e.g., Swanson et al 2015; Bubnicki et al., 2016; Krishnappa & Turner, 2014; Ivan and Newkirk, 2016; Young et al. 2018; Scotson et al. 2017; Reconyx, Inc., 2016; WildTrax 2019). Such camera trap software systems should, of course, include user interface features that encourage efficient human inspection of images and encoding of its data attributes. Those interface features should be based on a firm understanding of what analysts do when examining and encoding images, including mitigating human performance bottlenecks. However, most descriptions of these systems often provide only sparse details and discussions (if any) of the problems faced by analysts the system is trying to solve, the benefits of the particular interface features provided, and how the particular solution offered can be generalized to other systems. Thus there is a gap in how the lessons learnt from developing such systems could be applied to evaluate current software and/or developing next generation camera trap software interfaces. As a recent extensive World Wildlife Fund report on best practices for camera-trapping summarizes:

“Importantly, no single software package has emerged as a favourite amongst camera trappers, and lots of very different solutions to the problem of camera trap data management are currently being trialled... For any given camera trap project..., this makes it difficult to decide which software package to commit to. Many large camera trap projects ... have ended up designing their own systems from scratch. (p. 146)

The various software options available differ greatly in their approaches ...and you may need to test various options before deciding which one satisfies your requirements and most efficiently fits into your workflow (p.150).

(Wearn and Glover-Kapfer, 2017)

Our goal in this paper is to describe and explain *user interface design patterns* for software supporting how wildlife biologists perform camera trap image analysis during the tagging step. By way of background, the notion of design patterns were first introduced by architect Christopher Alexander, as a documented reusable and proven solution to commonly occurring architectural design problems (Alexander, 1977). Design patterns are typically derived by examining existing solutions to design problems (which may include ‘folk’ solutions) and generalizing them. Design patterns were later advocated as a way of describing common solutions to typical software engineering problems (Gamma et al, 1994), and to human-computer interaction problems (Borchers, 2001). Design patterns are usually structured as a name, a problem that explains it, and a design approach to a solution that solves the problem. Importantly, a design pattern is not a finished design. Rather, it is a description or template for how to solve a problem that can be used in many different situations.

We base our design patterns on both our understanding of what biologists require and the user interface features that support their needs from over eight years of development and application of the freely-available Timelapse Image Analysis system¹ (Greenberg and Godin, 2015; Greenberg 2019). For each design pattern presented, we describe the problem faced by the image analyst, a design approach that solves that problem, and a concrete example of how Timelapse addresses that design pattern².

2. THE DIVERSITY OF CAMERA TRAP RESEARCH PROJECTS, GOALS, AND IMAGES

Camera traps are used to address a wide diversity of ecological research and management objectives and associated taxa. This diversity leads to large differences in how cameras are situated and configured, the kinds of images collected, how analysts would examine those images, the attribute data recorded from images, and how the collected data would be subsequently analysed. For example, Figure 1 illustrates differences between two images of the same species with varying research objectives. In Figure 1a, the camera was set to motion-triggering to capture close-up views of mountain goats (*Oreamnos americanus*) as they passed by. Attributes of the goat could then be analyzed (e.g., species identification, sex, estimated age, individual identification, etc.). In Figure 1b, the camera was set to timelapse mode that took an image every 5 minutes in order to monitor the activity of a herd of goats over time on a distant pasture and mountain-side. Attributes of herd activity could then be analyzed (e.g., counts, duration in the meadow, ratio of kids to adults, etc.). Figure 1b includes a small herd in the green pasture, each goat just visible as white dots.

¹Timelapse availability is described at the end of this paper.

² Example videos illustrating some of the Timelapse features described in this paper can be found in:
<http://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications/Publications/2019-PityTheAnalyst.RemoteCameraWorkshop.pptx>



Perhaps the most familiar uses of camera traps and the subsequent analysis of its images are in some form of *in situ* wildlife monitoring (e.g., Steenweg et al., 2017, Burton et al. 2015). For example, in a review of 266 camera trap publications over a 6-year period, Burton et al. counted a range of ecological objectives and responses metrics including: relative abundance (43.6%), presence-absence (41.4%), behavior such as activity patterns and diet (32.3%), population density (15.8%) and occupancy (15.4%) (Burton et al., 2015, p. 678). Other examples include:

- monitor species diversity and inventories (e.g., Glover-Kapfer, et al., 2019; Ahumada et al., 2013; O'Brien & Kinnaird, 2010),
- measure population abundance, density, and distribution of marked and unmarked populations (e.g., Royle et al., 2018; Whittington et al., 2019; Karanth and Nichols 1998; Heilbrun et al. 2006; Goswami et al. 2007; Rowcliffe et al. 2008),
- examine multi-species dynamics (Swanson et al. 2015),
- estimate population trends (e.g., Karanth et al. 2006),
- correlating wildlife abundances to anthropological stressors (e.g., Fisher and Burton, 2018) including human activity, and to mitigation efforts (e.g., Whittington et al., 2019; Pollock et al., 2017),
- quantify animal behavior and success rates in DNA hair capture sites (Clevenger and Sawaya 2010),
- linking seasonal plant phenology to climate change and wildlife distributions (e.g., Laskin et al., 2019; Mills et al., 2018), and
- quantifying how seasonal changes in coat colour are influenced by climate change (e.g., Mills et al., 2018).

Camera traps can also help answer very specific research questions. For example, Garcia-Salgado et al. (2015) analyzed the diet of nesting raptors by examining images for prey deliveries to nests. Rollak et al. (2013) similarly deployed cameras around nests, but in this case to study the breeding behaviors of turkey vultures. Jumeau et al. (2017) used camera traps to estimate the effectiveness of wildlife crossing structures for small mammals.

Camera traps are also used to monitor and analyze human activity. Examples include analyzing images to: count the number of anglers actively fishing in order to estimate angling effort in small lakes fisheries (Greenberg and Godin, 2015); evaluate the influence of human disturbance on wildlife (e.g., Oberoslerab et al. 2017, Blake et al, 2017), quantify levels of human use and type of human activities (e.g., Fairfax et al. 2014; Campbell 2010); and detect types and occurrences of illegal activity in wildlife sanctuaries including identifying perpetrators (e.g., Hossain et al. 2016).

The above is just a very small sampling to indicate the diverse use of camera traps (e.g., see Wearn and Glover-Kapfer 2017 Section 5; Steenweg et al., 2017). While there may be some overlap in the kinds of images gathered across projects, we can easily expect differences between projects in: the kinds of images that are captured, the data attributes that would be encoded from these images, and the ways analysts would have to examine those images to extract and record that data. Our design patterns reflect this diversity of project objectives. Some design patterns are broadly applicable and useful in all kinds of image classification, while others may be of use in narrower suites of projects. More generally, software based on these design patterns will include tools for examining images, will provide a flexible user interface for efficiently encoding project-

specific attributes, and will simplify data entry by automatically extracting available image features as data. Collectively, it can substantially increase project efficiency, increase data quality and improve the user experience.

3. METHODOLOGY: DECONSTRUCTING THE DESIGN OF TIMELAPSE INTO DESIGN PATTERNS

Our methodology for understanding how analysts classify data and what user interface design patterns would be useful was derived from deconstructing our real-world experiences designing, implementing and refining the Timelapse open software system. Timelapse was specifically developed to help analysts inspect and classify camera trap images via tagging. Timelapse is conceptually simple: it displays images to analysts, along with a variety of project-specific (and user generated) attribute fields that they can efficiently fill in to describe the image. Yet its design goes far beyond that, for it includes many features addressing the subtleties of the analyst's workflow.

Timelapse evolved through many versions over its eight years. Its capabilities were designed to meet a broad variety of ecological needs as requested from a diverse international user community comprising different agencies (government, industry, university, independents) and biologists (researchers, practitioners, students). Most had differing projects and goals (e.g., wildlife monitoring, angling effort in fisheries, environmental monitoring, human monitoring, etc.). Our various redesigns of Timelapse were further informed by:

- ongoing discussions with both project managers and analysts whose work involved camera trap image analysis,
- feedback from analysts who had used Timelapse to inspect millions of images (e.g., problems, feature requests, bottlenecks), and
- observational studies of analysts using Timelapse as they did their work (e.g., Greenberg and Godin, 2015).

To provide context, Timelapse originated through a conversation between a Computer Science Professor specializing in Interaction Design (the 1st author), and a small lakes fisheries research biologist (the 2nd author). The biologist had described how her research team used camera traps to monitor ~75 lakes for fishing activities, and the difficulties, inefficiencies and expenses incurred when examining and encoding data across the hundreds of thousands of images (and more recently of videos) produced by those cameras. The computer scientist offered help, where he would design a system to ease that process. Using fairly standard techniques in the field of Human Computer Interaction (e.g., Shneiderman et al., 2016), the computer scientist observed and interviewed the technicians analyzing images, paying particular attention to their workflow and bottlenecks, and then translated those into a requirements analysis. Those requirements were then used to inform the design and implementation of the first version of Timelapse. Development did not stop there, where the computer scientist evolved its design further by observing how technicians used Timelapse in practice, and by discussing its use and limitations with the biologist and her technicians. These observations helped both biologist and the computer scientists understand and articulate the subtleties and variations in the workflow that arose due to various factors. For example, the type of images captured at particular sites differed considerably, which often led to workflow alterations in how technicians analyzed images. As well, particular sub-projects required different image features to be examined and encoded, again resulting in workflow differences.

As word of Timelapse spread, other researchers and managers from different domains and with different project needs began to use it. As with the small lakes fisheries team, those researchers discussed with the author how Timelapse was used by them, and how it could be improved to meet the demands of their own workflows. The author would then modify Timelapse to meet their needs. For example, the 3rd author of this paper is a wildlife ecologist who coordinates a vast camera network deployed across several Canadian national parks, with numerous image analysers, and a variety of research and monitoring projects. As Parks Canada began to use Timelapse, subsequent versions were influenced by discussions with the technicians responsible for classifying millions of images. The design of Timelapse thus evolved over eight years, informed by the input of diverse users across an international community. Importantly, the author followed a significant design constraint: Timelapse had to remain a tool usable by its broad community. That is, new interface features would be added only if they were potentially valuable to a broad range of projects and users, or – at the very least – could be ignored if they were not needed.

The remainder of this paper deconstructs and generalizes as design patterns the key workflow tasks and problems seen, and how they informed the corresponding Timelapse interface design solutions. Each design pattern begins with a short descriptive title, the problem faced by analysts, a general description of an interface design solution that mitigates that problem, and a concrete example of how Timelapse realized that design pattern. We also invite the reader to run Timelapse as they read the design patterns, to help them better understand the nuances of the proposed solution.

While we highlight how Timelapse instantiates a particular design pattern solution, we stress that the design pattern is more general than that, as it can also inform alternate software designs. For example, designers can selectively incorporate particular design solutions as seen in Timelapse into their own systems. Alternately, designers can use our design pattern problem descriptions and general solutions to create their own novel, alternate ways to solve those problems. Finally, project

managers can match design patterns against their project needs, and then evaluate competing software solutions to see if they have interface features that support the design patterns relevant to their project.

We recognize that other camera trap software systems may offer similar or alternate solutions to our design patterns. However, we restrict our example solutions to Timelapse, as a comparative review and analysis is beyond the scope of this paper. Still, the design patterns supplied here should allow readers to reconsider if and how other software solutions address particular design problems, and may even help them identify problems and design patterns that are outside of what we provide below.

4. TERMINOLOGY.

The terminology applied to the use of camera traps varies considerably. This is likely due to its relatively recent uptake, and its diversity of applications across a broad variety of disciplines and projects. For ease of explanation, we define and explain a few terms here.

- *Camera traps* are special purpose camera built into a rugged (and sometimes camouflaged) weather-proof housing (e.g., Swann et al., 2010). They are typically mounted on a fixed object such as a tree to capture a desired field of view. They may be left in place for several months to years. As previously explained, most can operate in either *motion-triggered*, or *timelapse* mode. For night-time shots, some cameras include either an infra-red (non-visible) or incandescent (visible) flash. Some cameras burn extra information onto the image and/or record extra information in the image metadata (e.g., date, time, temperature, barometric pressure). They can also include video capabilities and high resolution panoramas.
- *Station* is the location of a particular camera (Tobler et al., 2015).
- *Study area* is the geographic region covered by a set of camera traps.
- *Field of view* is the area recorded on the camera images.
- *Servicing period* is the duration between servicing a camera, i.e., placing a new SD card into the camera, and then retrieving the images on that card sometimes later on.
- *Session* (sometimes called *season*) is the duration of the sampling period where the cameras were active (Tobler et al., 2015).
- *Image set* is the *collection* of images being analyzed by an analyst. The images that comprise an image set depends on how the project manager conceptualizes and manages images as a set. As one example, an image set can comprise only the images just extracted from the SD card at a particular station at the end of a servicing period. Alternately, an image set can comprise all images at a particular station (i.e., the entire session), where new images extracted from a servicing period are added to the collection of older (perhaps already analyzed) images. As a further example, an image set can comprise all images taken from multiple stations at a particular study site.
- *Analyst*. Person who examines each image enters attribute data associated with that image.

5. THEME: DATA CONSISTENCY

The ultimate goal of the analyst is to enter attribute data that reflects the contents of the images. Statistical analysis of attribute data from an image set typically occurs at a later date. The value of attribute data depends upon its consistency (explained below) and quality. Thus, data entry protocols for what attributes to collect must be developed prior to image classification.

Design pattern: Specify and deploy a common data schema.

Problem - The data required and how they are named may be inconsistent between analysts. Projects typically involve multiple cameras located at multiple stations at one or more study sites. In turn, this can generate a large number of distinct image sets, each containing many thousands of images. Multiple analysts may be involved (perhaps including volunteer citizen scientists), each analyzing the images within a particular image set. A key issue is maintaining data consistency across different image sets and different analysts, i.e., where all analysts are inputting consistently formatted data into a set of consistently named data fields. Without data consistency (e.g., if each analyst idiosyncratically specified what data should be encoded from images, in what format, and under what name) it would be extremely difficult to make sense of the data across analysts and image sets.

Design approach. The project manager should initially decide what data should be collected from the image sets and communicate those needs to the analysts as a standardized *schema* that defines and specifies the data of interest. Analysis of image sets by all analysts should then be based on that schema. The schema should define the required data fields, how they are named, their data type, their format, and constraints on what values they allow.

Timelapse example. Project managers use a Timelapse utility called the *template editor* (Figure 2) to construct a template file that defines the data schema. Analysts place the data template file in the folder containing the image set. When the actual Timelapse application is opened (Figure 3), it uses the template schema to build the user interface and to construct the database table that will ultimately contain the data entered by the analyst. Figure 2 illustrates a screen shot of the template editor in action, where the project manager has specified a schema³. Each row specifies all attributes of a particular data field, while each column names the attribute. The *template editor* allows project managers to easily create project specific schemas. Each attribute field of the schema is developed with the following options:

- **Type** indicates the data type and its format. For example, the type *DateTime* and *UTCOffset* follow the international standard for specifying dates. *Counters* are positive integers. *Flags* can only contain true/false values. *Notes* are free-form text fields. *Choices* are constrained to a limited number of possible values provided by the project manager. For example, a field labelled ‘Weather’ could be constrained to the values ‘Sunny’, ‘Cloudy’, ‘Foggy’ etc.
- **Data Label** names the data field containing that data, i.e., the column name as it would appear in a database table or a spreadsheet.
- **Default Value** indicates the initial value of that data, which will be applied to every image seen by Timelapse.
- **List** defines the allowable entries for Choice data types. Selecting ‘Define list’ raises a small window where the manager can type in those entries (see Figure 2, lower right side).

The screenshot shows the 'Timelapse Template Editor (TimelapseTemplate.tdb)' window. It has a menu bar (File, View, Help) and a toolbar (Brief Instructions, Template). The main area contains a table with columns: Type, Default Value, Label, Data Label, Tooltip, Width, Copyable, Visible, and List. The table lists various attributes like File, RelativePath, Folder, DateTime, UtcOffset, ImageQuality, DeleteFlag, Counter, FixedChoice, Note, and Flag. To the right of the table are buttons: Add, Count, Choice, Note, Flag, and Remove. A 'Define choice menu' dialog is open, showing a list of choices: Sunny, Cloudy, Foggy, Raining, Snowing, and an option to 'Include an 'empty' item'. Below the table, there is a section titled 'What the interface will (roughly) look like. Drag and drop these controls (by their labels) to re-arrange their order to your preference'. This section shows a preview of the user interface with fields for File, RelativePath, Folder, DateTime, Image Quality, Delete?, Goats, Hikers, Weather, Temperature, Analyst, Comments, and Publicity?. Below this is another section titled 'The spreadsheet column order. Drag and drop columns to re-arrange their order to your preference', showing a list of columns: File, RelativePath, Folder, Date, Time, ImageQuality, DeleteFlag, GoatCnt, HikerCnt, Weather, TempC, Analyst, Comments, and PublicityShot.

Type	Default Value	Label	Data Label	Tooltip	Width	Copyable	Visible	List
File		File	File	The file name	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
RelativePath		RelativePath	RelativePath	Path from the folder containing the template and image data files to the file	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Folder		Folder	Folder	Name of the folder originally containing the template and image data files	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
DateTime	1900-01-01T12:00:00.000Z	DateTime	DateTime	Date and time taken (Year-Month-Day Hours:Minutes:Seconds:Milliseconds)	160	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
UtcOffset	0.00	UtcOffset	UtcOffset	Universal Time offset of the time zone for date and time taken	60	<input type="checkbox"/>	<input type="checkbox"/>	
ImageQuality		Image Quality	ImageQuality	System-determined image quality: Ok, dark if mostly black, corrupted if it ca	85	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Define List
DeleteFlag	false	Delete?	DeleteFlag	Mark a file as one to be deleted. You can then confirm deletion through the	20	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Counter		0 Goats	GoatCnt	A count of how many goats appear in this image	30	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Counter		0 Hikers	HikerCnt	A count of how many hikers appear in this image	30	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
FixedChoice		Weather	Weather	Current weather conditions	100	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Define List
Note		Temperature	TempC	Temperature in Celsius	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Note		Analyst	Analyst	Name of the analyst coding these images	100	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Note		Comments	Comments	Other thoughts about this image	100	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Flag	false	Publicity?	PublicityShot	This image as useful for publicity purposes	20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Define choice menu, with 1 item per line

Sunny
Cloudy
Foggy
Raining
Snowing

☒ Include an 'empty' item

Cancel Okay

What the interface will (roughly) look like. Drag and drop these controls (by their labels) to re-arrange their order to your preference

File RelativePath Folder DateTime 01-Jan-1900 12:00:00 Image Quality Ok Delete? ☐

Goats 0 Hikers 0 Weather Sunny Temperature Analyst Comments Publicity? ☐

A count of how many goats appear in this image

The spreadsheet column order. Drag and drop columns to re-arrange their order to your preference

File RelativePath Folder Date Time ImageQuality DeleteFlag GoatCnt HikerCnt Weather TempC Analyst Comments PublicityShot

Figure 2. The Template Editor. The project manager defines the attribute data of interest (the schema) as well as the associated user interface specifications by form-filling (top pane). The middle pane generates a preview of the user interface data entry controls that will be seen by the analyst. The bottom pane shows how the data of interest will be stored as columns within a database table.

³ This paper uses a deliberately simple schema / template for illustration. In practice, schemas can be quite rich and complex

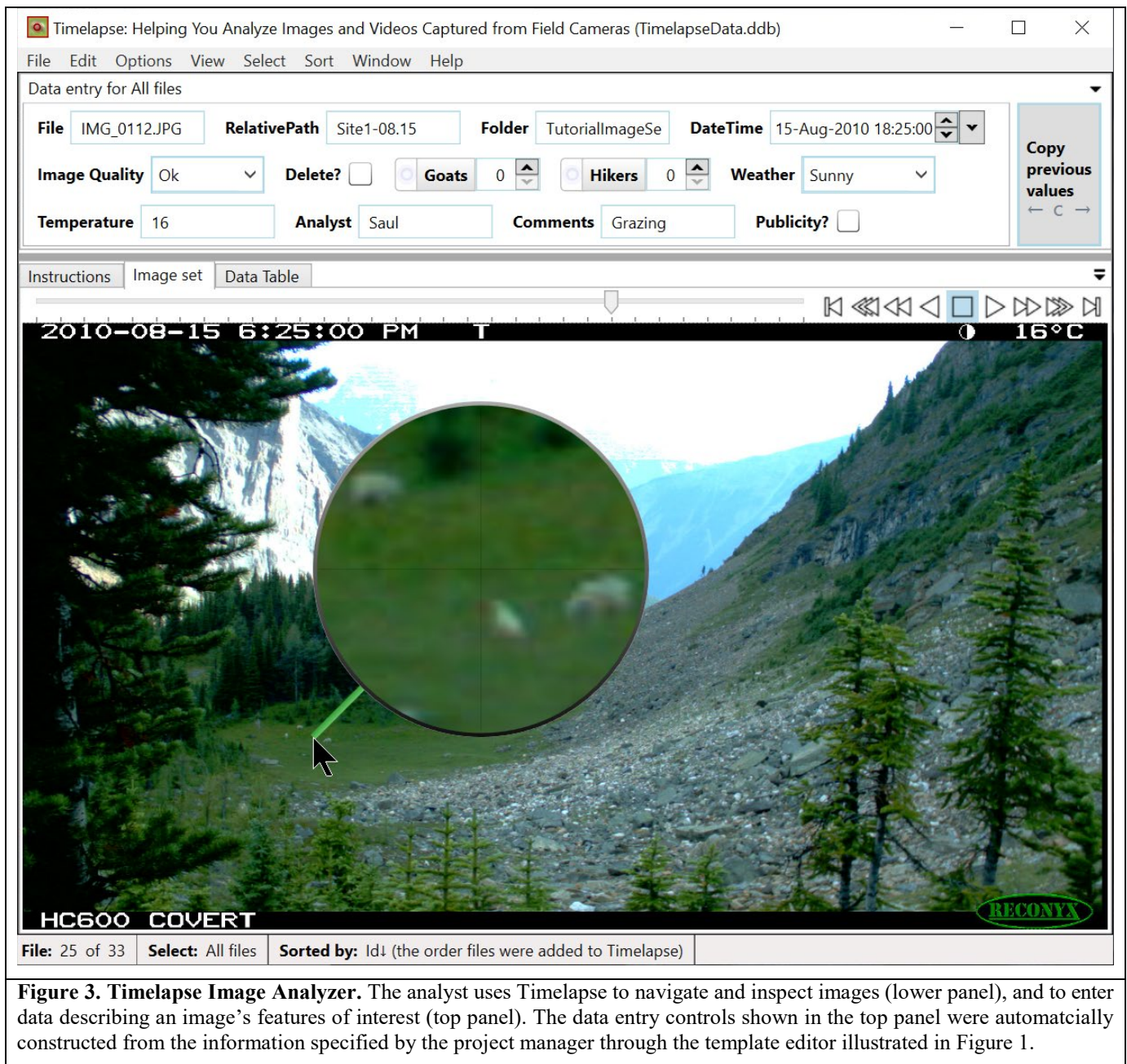


Figure 3. Timelapse Image Analyzer. The analyst uses Timelapse to navigate and inspect images (lower panel), and to enter data describing an image's features of interest (top panel). The data entry controls shown in the top panel were automatically constructed from the information specified by the project manager through the template editor illustrated in Figure 1.

Design pattern: Present the data schema in the analyst's language.

Problem - Data schema terminology may not be in the analyst's language. Computer systems often ask users to enter data by either filling in rows in a table, or via data entry widgets (textboxes for text entry, menus for selecting choices, etc.). These are often labelled in some manner (e.g., column names in a table or a name associated with the data entry widget). Labels are important, as analysts need to understand what data they are expected to enter. The issue is that the terminology used may not be in the analyst's language. For example, systems may use the database field name to label a data entry widget, but these names may not be understandable (e.g., acronyms, abbreviations, technical terms, ambiguous meanings, etc.) or insufficient to describe what the analyst should enter. This can lead to inconsistencies between the project manager's expectations of the data required vs. the analysts' interpretation of what data should be entered. A related issue is that the schema may include fields that are of little interest to the analyzer, such as fields they are not expected to review or fill in. Their inclusion in the analyst's user interface would add clutter and perhaps confusion, or have the analyst fill in fields unnecessarily. Finally, because the project manager and all analysts need to communicate to one another, the terminology used to identify that data

should be common. These terminology problems are particularly endemic to analysts who inspect images only occasionally, as they may forget what particular terms mean, and to citizen scientists with minimal training.

Design approach. The data-entry interface presented to the analyst should be expressed clearly in familiar words, phrases, concepts and explanations rather than in system-oriented terms. Non-relevant data items should be removed (Nielsen, 1993, Chapter 5). One way to do this is to associate the terms in the data schema with a more understandable terminology and descriptions of what data they are expected to enter, and whether or not particular data items should be displayed to the analyst. While the interface seen by the analyst would be constructed using that terminology, the data entered by the analyst would be stored under the data labels provided by the data schema, thereby maintaining data consistency.

Timelapse example. Using the *Timelapse Template Editor*, the project manager specifies the terminology of data entry controls, i.e., the interface controls associated with every data field used by the analyst to enter data. For example, consider the data field with the data label of 'GoatCnt' in Figure 2. In the *Label* column, the manager specified 'Goats' as a more human-readable alternative to that data field. In the *Tooltip* column, the manager provided a brief help explanation of what should be entered: 'A count of how many goats appear in this image'. By unchecking the 'Visible' checkbox, the manager has indicated that the 'UtcOffset' data field should not be displayed to the analyst. As the manager performs these actions, a live preview of the user interface (Figure 2, middle pane) reflects the actual user interface that will be seen by the analyst when using the Timelapse system (Figure 3, top pane). Both illustrate how the data controls adopt the analyst-oriented terminology and explanations specified in the template (e.g., the 'Goats' control and the displayed tooltip). When the analyst enters data in that control (Figure 3), it is stored in the corresponding data field (e.g., 'GoatCnt').

Design pattern: Data entry controls should minimize input errors by constraining input to the data field's type.

Problem - Data input errors are commonplace. When analysts enter data, they may inadvertently introduce errors into the stored data. For example, the entered data may be outside of what is expected (e.g., non-numeric characters entered into an integer data field, Yes/No instead of True/False as expected by a boolean data field). As another example, the entered data may not be in the correct format (e.g., Date may be incorrectly entered as 'mm/dd/yyyy' order instead of the expected 'dd/mm/yyyy').

Design approach. Input controls should provide visual hints of what input is accepted, and should only accept input that matches the data field's type and format. This pattern is now common in many modern user interfaces, where myriads of input controls suggest and constrain what can be entered into them.

Timelapse example. Timelapse generates its user interface from the description provided in the template editor (Figure 2), where each data entry control is based on its corresponding template specification (Figure 3, top pane). The data entry control constrains the input, where only valid data can be entered into it. For example, an analyst interacting with the DateTime data entry control (Figure 3, top pane, right side) can only enter or edit valid date and time values. Counter controls (e.g., Goats, Hikers) only accept positive integers (either by typing or by mouse interaction). Choice data controls (e.g., Image Quality, Weather) restrict the analyst to selecting from a pop-up menu containing only valid text entries. Flags (e.g., Delete?, Publicity?) present themselves as a checkbox, where the control itself translates its 'checked' state to the two allowable data storage values of 'true' or 'false'. In all cases, the analyst's data entry is constrained by the corresponding data control to allow only legal values, which reduces possible input errors significantly.

6. THEME: INSPECTING IMAGE FEATURES

A major part of the analyst's tasks is to inspect the image to discover (and record as data) features of interest. Yet inspection can be problematic for some images, especially when the features of interest are not discernable at a glance.

Design pattern: Allow the analyst to examine image details through magnification.

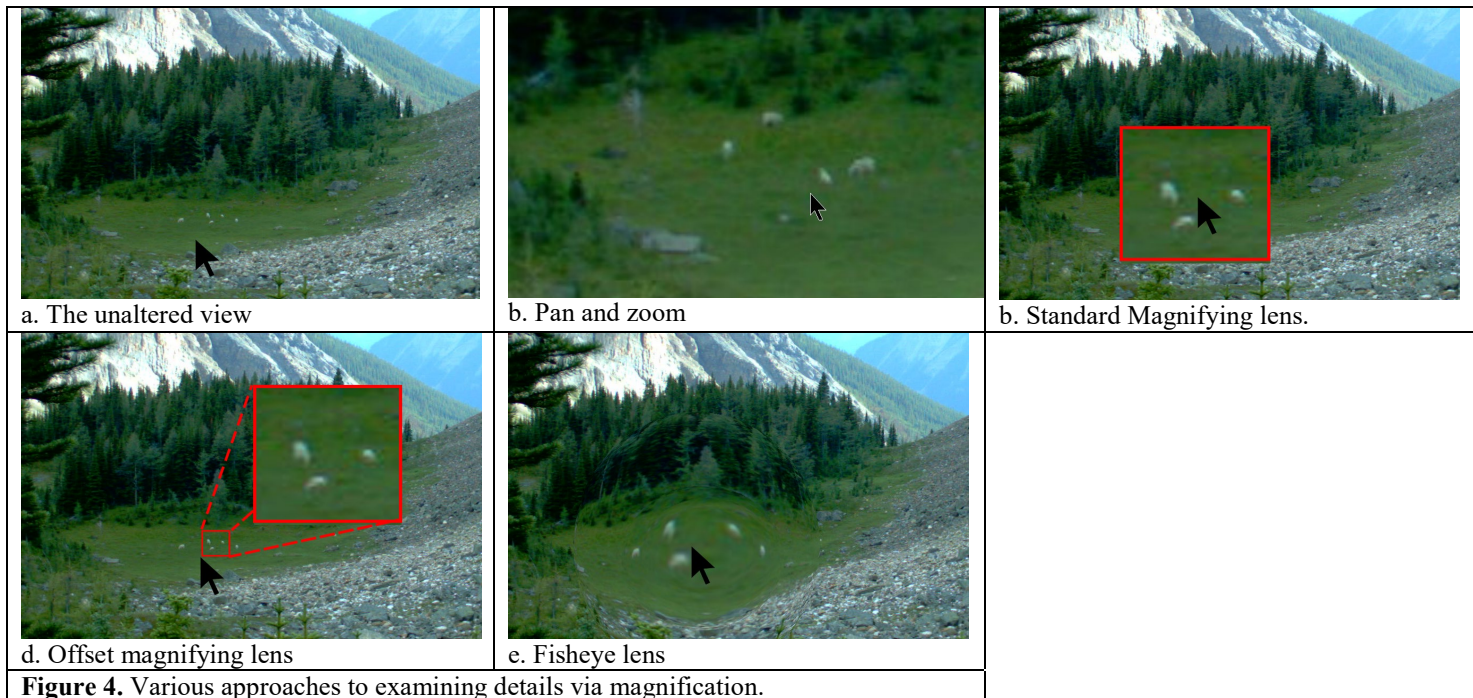
Problem - Relevant image details may be difficult to examine. Depending on what is captured in the image's field of view, analysts may have to inspect small image details in order to classify what is there. One example are camera images that capture large fields of view, where the items of interest are very small. Figures 1b, 3 and 4a illustrate such an image, where the camera is oriented to capture distant goats as they wander through a mountain side and meadow. Other examples include cases where the analyst has to identify small animals, or details of that animal (e.g., sex, health), or where the image only displays a portion of that animal (e.g., because of occlusion by vegetation, or because the animal is only partly in the field of view).

Design approach. Image magnification can help the analyst examine small image features. However, due to the number of images inspected, the analyst's interaction with the provided magnification technique must be efficient to use. Various

magnification interaction techniques are known. The most common is perhaps a *pan and zoom* facility (Figure 4b), which allows the analyst to zoom (magnify) a particular image region (called the *focus*). While powerful, zooming into local detail incurs the cost of losing *global context*, i.e., only the zoomed in portion of the image is visible, which means informative details outside of the focus cannot be seen at the same time.

Another common approach mimics a *magnifying lens* (see Figure 4c), where the area under the cursor (the focus point) is overlaid by a small zoomed in region. The area being magnified is immediately updated as the cursor is moved. Again there is a tradeoff: while the magnified areas shows the focus area under the cursor, it occludes some of the surrounding global context as it overlays it. For example, Figure 4c magnifies 3 of the goats, but at the cost of occluding the other two nearby goats.

Far more sophisticated magnification approaches have been developed in the field of *information visualization* (e.g., Spence, 2014, Shneiderman et al., 2016 Ch. 12). For example, an *offset magnifying lens* avoids occlusion by offsetting the magnified area away from the cursor's focus area (Ware and Lewis, 1995). As Figure 4d illustrates, the small square by the cursor is the area to magnify, where that zoomed in area is shown offset in the larger square. Thus the analyst can simultaneously see both the unzoomed and zoomed in area at the same time. Another approach is *focus plus context magnification*. For example, a *fisheye lens* distorts an image to provide magnification in place (Carpendale et. al., 2004). As Figure 4e illustrates, the highest magnification is focused under the cursor, where a drop-off function applies progressively less magnification away from the cursor. The advantage, as seen in Figure 4e, is that local detail is shown in place within the global context. This also avoids separation of the magnified vs. unmagnified image as evident in Figures 4b+c.



Of course, the value of magnification is affected by image fidelity. Some systems may, for example, reduce an image's display resolution for performance purposes with the side effect of comprising image details. The fewer the pixels read in, the less memory required and the faster images can be displayed. The tradeoff is that magnification would then produce 'fat pixels' (*aka* blurry images) rather than details. Fortunately, various image processing techniques are known that can efficiently display the whole image at low fidelity, while reading in high resolution details for only the magnified regions (Carpendale et. al., 2004).

Timelapse example. Timelapse contains several methods for rapidly examining image details through magnification.

- a. **Zoom and pan.** The analyst can zoom and pan into any part of the image using the scroll wheel and the mouse. Zooming occurs at the cursor location, while panning to a particular image region is done by dragging the image with the mouse. Timelapse's zooming and panning features also include nuances that support how an analyst would use it over multiple images. First, the analyst can 'bookmark' a particular zoomed in area, where she can flip between the zoomed and normal

image with a single keypress, thus maintaining some sense of how the zoomed-in focus relates to the global context. Bookmarks have other advantages. For example, if the analyst was interested in animal activity in the pasture seen in Figure 3, she could zoom into that pasture and bookmark it. When checking other images for activity in that pasture, she can use that bookmark to zoom into the same corresponding area. Second, zoom/pan levels are maintained when navigating between images. For example, the analyst could zoom into the pasture of Figure 3 and then see how the goats have moved around that pasture by navigating to the next few images.

- b. **Offset magnifying lens.** Similar to Figure 4d but with different visuals, the analyst can turn on a magnifier that displays zoomed-in image of the area around the cursor: the lens is offset to avoid occluding that area. Figure 3 illustrates the Timelapse magnifier in action, where the analyst is using it to detect and examine a herd of goats. The analyzer can easily scan the image details for features of interest by dragging the magnifier, whose magnified content is instantly updated. The analyst can also quickly adjust the amount of magnification through a few keystrokes.

Design pattern: Enhance the noticeability of small entities within images.

Problem - The presence of small entities may be difficult to notice. Various projects use cameras in *timelapse* mode, where periodically taken images capture a very wide field of view. We already saw how Figures 1b and 3 illustrate one actual example, where the camera was located to capture distant goat activity in a pasture and on a mountain-side. Another real example includes cameras positioned to capture distant anglers on and around a large lake area (Greenberg & Godin, 2015). The issue is that analysts may not notice the presence of these small entities. This becomes more problematic when a run of images being examined have nothing in them, as analysts expect that pattern to continue. Magnification, while somewhat helpful, is best used to examine details *after* an entity has been noticed.

Design approach. Various techniques can enhance how the analyst can notice small entities in a scene by making them visually distinctive.

Animation of image sequences visually highlights changes that occur by rapidly switching between images. Because the background scene is reasonably constant, the appearance, disappearance and movements of entities within the scene are often very noticeable.

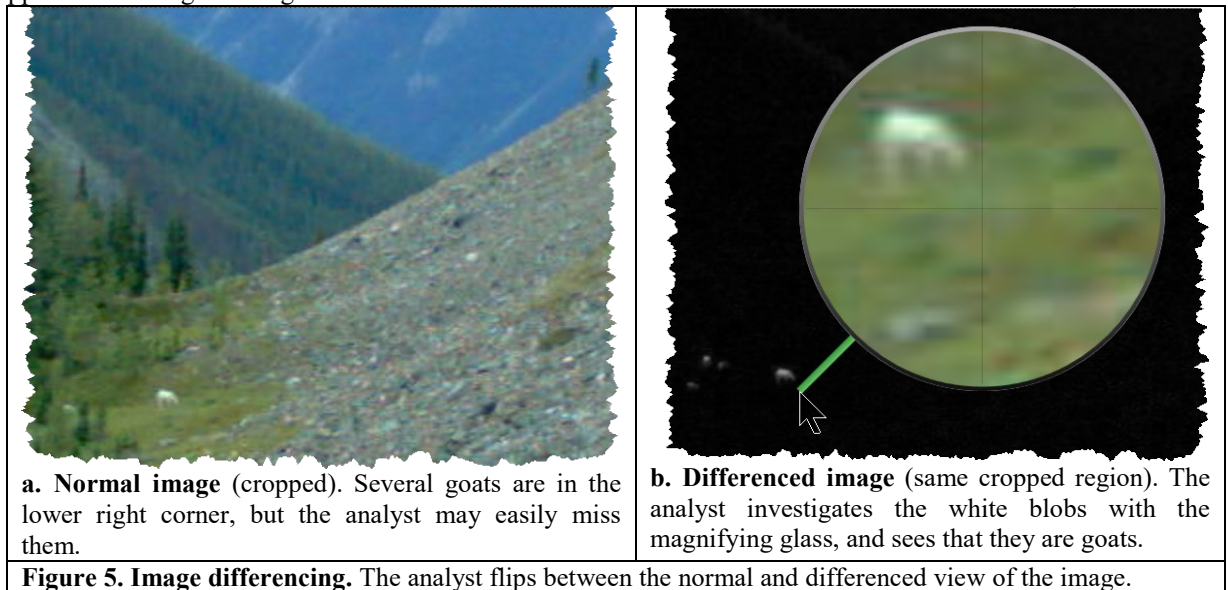
Image processing through image differencing compares, pixel by pixel, the current image against the previous and/or next image. A new image is generated from that comparison, where (for example) a white pixel is drawn if the compared pixels differ significantly in brightness and color (set by threshold values), and black otherwise. The resulting image visually highlights the differences in white, while removing the somewhat static background. Because entities appear, disappear, and move around a scene, the differenced image will display that entity as a white blob (usually in the shape of the desired entity) against a black background. Other image processing techniques may also help, such as motion tracking that track the position of an object over subsequent frames. We note that the effectiveness of image processing techniques can be compromised when large visual differences occur between the images, such as dramatic changes in image lighting, motion of nearby grass and branches affected by the wind, and even slight changes of a camera's position (e.g., due to wind effects on the tree it is mounted on).

Image enhancement. Many off-the-shelf photo viewing systems now include various ways to adjust an image. Examples include contrast adjustment, saturation and luminance of particular colors, dehazing, sharpening, edge detection, etc. In the image analysis context, an analyst could apply various adjustments on a test image and – if effective at enhancing an entity's visibility – have that setting automatically applied when viewing other images.

Timelapse example. Timelapse contains the first two methods above for enhancing the noticeability of small entities, both based on analyzing the differences between images. Timelapse does not include other image enhancement methods, but they could be added easily.

- a. **Animation through rapid image switching.** Timelapse lets the analyst rapidly switch between the current image of interest and the next or previous image (using the arrow keys), where images are displayed immediately and without flicker. The analyst perceives this as an animation, where the differences between images – such as a small moving animal – ‘pop out’. Furthermore, the magnifier and zoom/pan level are maintained at their current setting and position during image switching, which helps the analyst spot differences in a magnified region.
- b. **Image processing through image differencing** compares the original image to the previous image, the next image, or to both. The analyst toggles between the differenced and original image with a single key press. When blobs of interest appear, the analyst can use the magnifying glass (which displays that region in its original form) to investigate. Alternately, she can use rapid image switching to see if the blob has moved. Figure 5 provides an example of how this appears in practice. Figure 5a displays the normal image: the small goat can be easily missed or mistaken for a rock.

Figure 5b is the differenced image: not only is the goat highlighted as a blob, but another goat partly hidden in the trees on its left is revealed. Figure 5b also shows the analyst furthering inspecting a blob via the magnifier, which displays the goat as it appears in the original image.



7. THEME: NAVIGATING IMAGES

Analysts are often tasked with inspecting tens and even hundreds of thousands of images in an image set. Thus reviewing, searching and navigating between images should be rapid.

Design pattern: Provide tools that allow rapid navigation and review of images.

Problem - Tedious image navigation and review. An analyst may want to rapidly navigate and review a sequence of images for various reasons. She may want to scan all images quickly before coding them, in order to get a sense of what is in them. She may want to quickly move over ‘empty images’ (e.g., scenes with no wildlife in it) until she spots an image containing something of interest. She may want to visually search the image set for a particular scene, for example, an image with wolves and cubs. She may also want to search for a particular image by its file name. The problem is that image packages often differ considerably in how they support navigation, where some navigational methods can interfere with the analyst’s task. For example, image review would be severely impeded if each image has to be separately opened in its own window.

Design approach. Analyst often examine images sequentially to see how they unfold over time. Stepping forward and backward through them should be visually instantaneous and should require minimal effort, e.g., via a single key press or mouse click. Because image sets can number in the hundreds of thousands, analysts should be able to move, scrub and jump through images quickly, similar to how one can scrub through a video. To help the analyst visually review and compare images during navigation (such as to detect changes as discussed in the previous design pattern), display settings such as zoom levels and the image location on the screen should be kept constant.

Timelapse example. Timelapse contains many navigation methods, each allowing rapid image review.

- Forwards/Backwards controls.** Timelapse lets the analyst rapidly move either backwards or forwards between images via the keyboard (the arrow keys) or by the File Player (described next). Holding down the arrow key scrubs through successive images. Settings on the current image – the location of the magnifying glass, zoom and pan levels, image differencing (if any) are all retained during navigation,

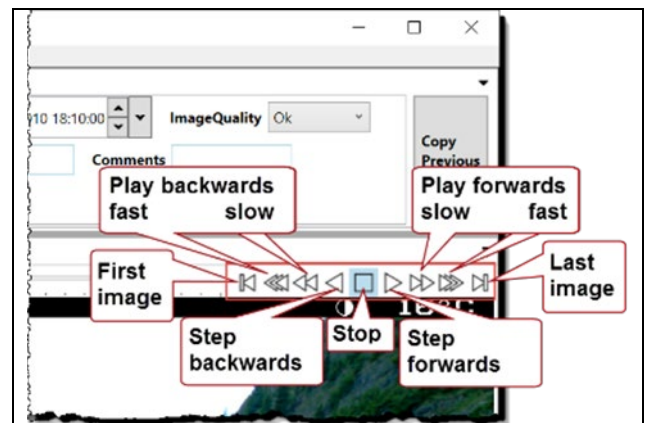


Figure 6. The Timelapse File Player.

allowing the analyst to rapidly compare images for similarities and differences as he or she views them.

- b. **File Player** (seen at the upper right of Figure 3 and annotated in Figure 6) provides an alternative mouse-operated navigational control. Depending on the button pressed, the analyst can step through images, jump to the first or last image, or automatically play (and thus review) successive images at slow and fast speeds. These speeds are user-configurable.
- c. **Navigational Slider** (next to the File Player, see top middle of Figure 3) allows the analyst to both scrub through and to rapidly jump across many images. Intervening images are displayed as fast as possible as the analyst moves the slider.
- d. **The Overview.** Analysts can ‘zoom out’ to see an overview containing multiple images, as illustrated in Figure 7. The more one zooms out, the more images are displayed, albeit at progressively smaller sizes. The analyst can navigate to a full sized view of a desired image (as in Figure 3) by clicking its thumbnail in the overview. The behavior of the navigational controls described above are also transformed to work with the overview. For example, the File Player controls now allow the analyst to navigate through successive images one by one, row by row, or page by page. Using the overview, the analyst can navigate and review collections of multiple images quickly.
- e. **Find Search Bar,** illustrated at the top left of Figure 7, is somewhat similar to search bars seen in text editors. The analyst uses it to find and display the next file in the image sequence whose file name partially matches the entered text. Figure 7 illustrates a search for any file name containing ‘05’. Find works in both the single image view (Figure 3), and in the overview where the found image becomes the first image in the displayed image array (Figure 7). Find also works on suffixes. For example, if an image set is interspersed with video files, searching on ‘.avi’ will step through all videos.
- f. **Navigating via the data table.** Analysts have the option of a database view, which displays all the data entered so far as a scrollable table. This is available through the ‘Data Table’ tab as seen in Figure 8. Each row represents all the data currently associated for an image. The analyst can inspect the rows for data of interest, and click on that row to navigate to and view the image associated with that row (akin to the display in Figure 3).

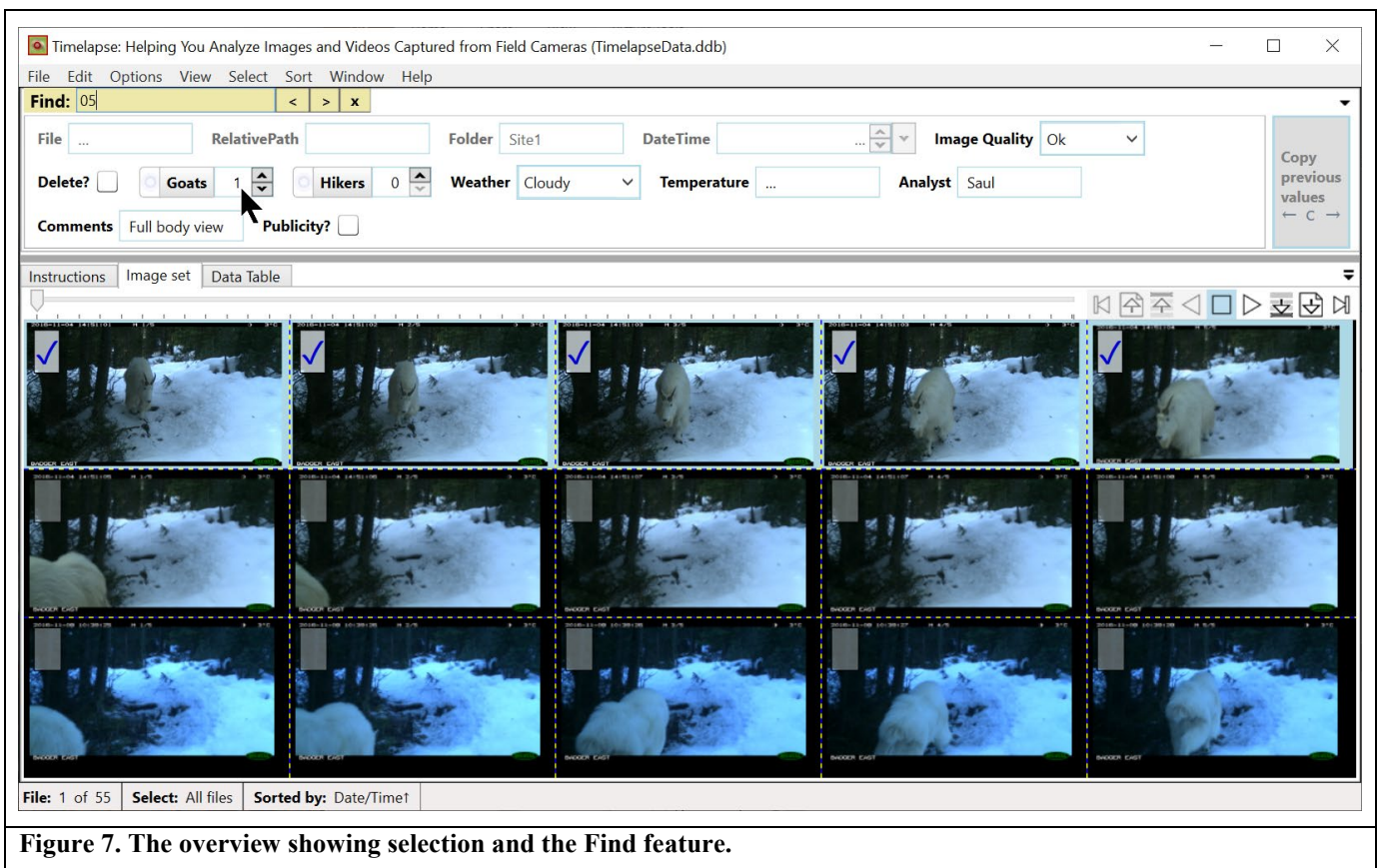


Figure 7. The overview showing selection and the Find feature.

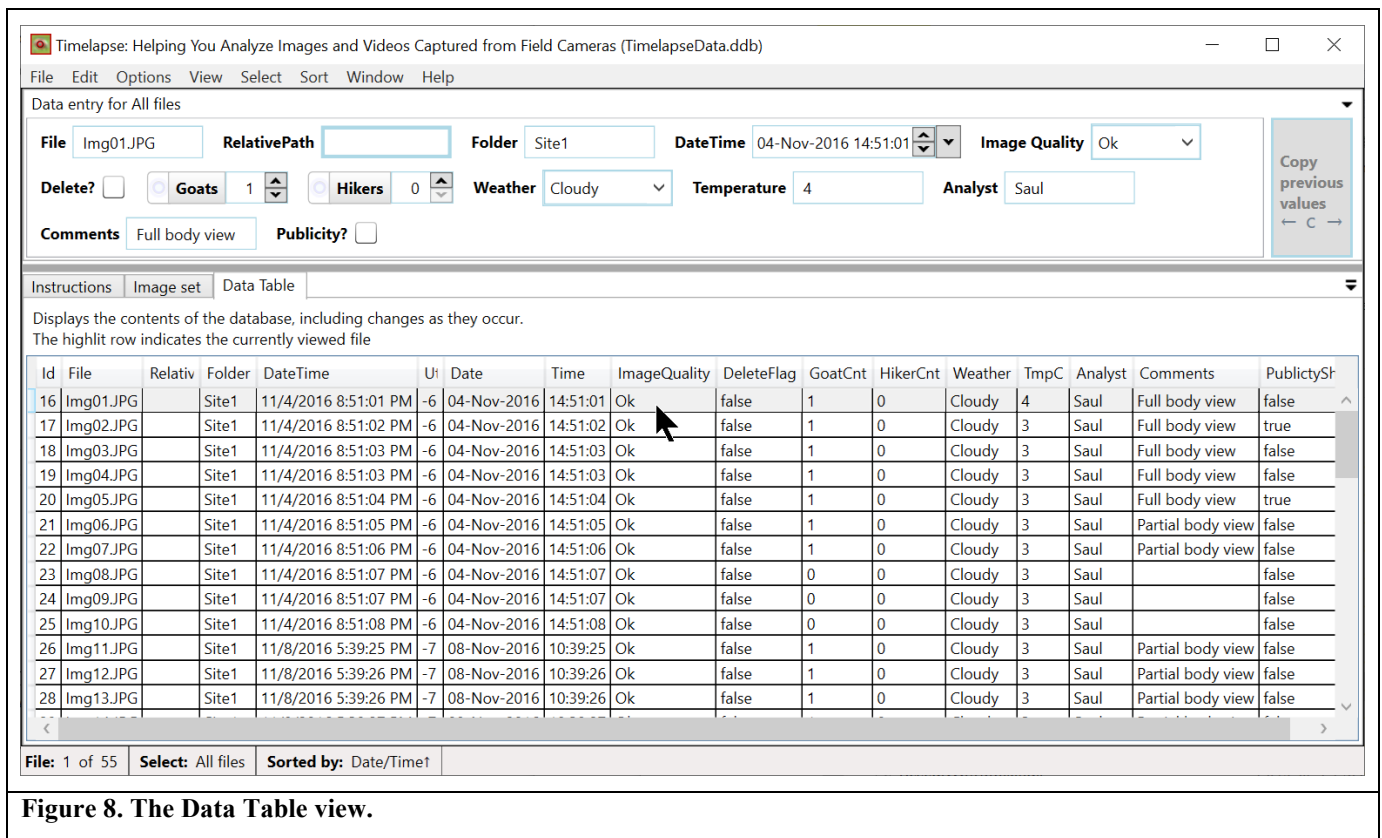


Figure 8. The Data Table view.

8. THEME: ENTERING DATA

Our example data schema illustrated in Figure 2 and composed as a data entry interface in Figure 3 has relatively few data entry fields. This contrasts with the actual number of data fields that analysts can encounter in practice. For example, one of the agencies using Timelapse composed and regularly used a template defining 30 separate data entry fields that analysts had to fill in. Even if only a subset of those fields relevant to a particular image had to be filled in, data entry can quickly become tedious, error prone, and very time-consuming when done over hundreds of thousands of images.

Design pattern: Data entry controls should minimize or eliminate typing when possible.

Problem - Typing is time-consuming and error-prone. Filling in data fields by typing is tedious. Mis-typing is common, and introduces errors and inconsistencies in the data.

Design approach. Selection (via the mouse or via tab/select/enter) should replace typing whenever possible. Since much data entry is repetitious, previously typed-in entries should be offered as candidates for selection rather than requiring re-entry.

Timelapse example. Several data controls available through Timelapse (e.g., see top of Figure 3) favor selection via the mouse or through the keyboard's tab and arrow keys. An analyst selects a Flag's true or false value by clicking on its checkbox. She selects from a Choice's limited possibilities via a pop-up menu. She can fill in Counters by clicking its up/down arrow buttons, or by clicking an entity in the image to count it (discussed shortly). She can edit the dates and times in the DateTime control by its up/down arrow buttons, or by directly selecting a date from a calendar. She can accept text predictions in Notes instead of typing an entry in full. Each Note tracks all previously typed text entries, and uses those to predict the rest of the text as the analyst types. For example, Figure 9 illustrates the text prediction that appears after the analyst has typed the single letter 'O'.

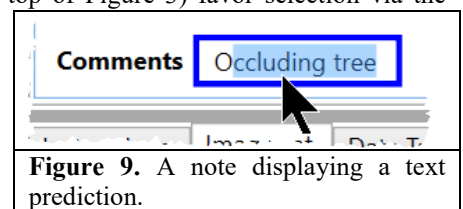


Figure 9. A note displaying a text prediction.

Design pattern: The system should allow one to visually mark the entities present in an image that have been counted along with its type.

Problem - Counting is difficult and error-prone when there are many countable entities present in an image. As previously discussed, some cameras are positioned to capture a wide field of view. In turn, the resulting images can contain many entities, perhaps of different types, that must be counted (e.g., Figure 1b). A wildlife monitoring example is a herd of animals present in the field of view, while a fisheries example is many anglers and non-anglers present on a popular lake's shoreline, or in boats (Greenberg & Godin, 2015). All entities must be categorized and counted. The problem is that mis-counting is easy. Common errors include losing track of the current count number, double-counting that counts an entity more than once, and omission errors where an entity is accidentally skipped.

Design approach. Visually marking entities as the analyst counts them can mitigate common counting errors: the analyst can discern what has been counted and what is yet to be counted. If different entities are present and being counted, the visual mark could also indicate how that entity was identified. Visual marking also affords validation, where a (perhaps different) analyst can later review the image and its marked entities for counting or classification errors.

Timelapse example. The Counter data entry control supports interactive counting and visual marking, and is illustrated in Figure 10. Here, the analyst has activated the 'Goats' interactive counting mode by clicking its radio button. The analyst then counts goats simply by clicking next to each one: each click increments the count and adds a colored marker at that spot. Markers also work with the magnifying glass, where the analyst can inspect entities before marking them. Finally, markers provide feedback as to which Counter button they are associated with. For example, hovering over a marker reveals that it was counted as a 'Goat' (as in Figure 10). Conversely, hovering over the Goats Counter button will highlight only those marks in the image counted as a 'Goat'.



Figure 10. An activated Count control showing the visual marks next to the counted goats.

Design pattern: The system should automatically fill in data fields if the information is available.

Problem – The analyst has to manually re-entering image data even when it is available in a computer-readable form. Analysts find it particularly frustrating when they have to re-enter information that is already available electronically. This problem usually arises when software does not try to read in that information, or cannot make sense of that information without some guidance.

Design approach. The system should try to automatically fill-in useful and readily available known information. This can include 'standard' information such as file names, file location in folder, and the date and time the image was taken. As well, image files typically contain embedded metadata that describes attributes of the image, where some of these fields could be of interest and automatically imported. Yet metadata introduces its own problems. Most camera vendors embed a mix of standard and non-standard (proprietary) metadata, which means that the information available is highly camera-dependent. For example, some may include ambient temperature and GPS location of the station, but others may not. Another issue is that different vendors may name fields differently, e.g., the outside temperature may be recorded in one camera as 'Ambient temperature', and in another as 'Temperature C'. Thus the analyst should be able to specify what metadata fields of interest, if any, should be imported, and where that information should go.

Timelapse example. Timelapse automatically fills in data fields in two ways.

- Standard file information.** Timelapse template schemas always include several default data fields representing standard file information: its name, its location (as a folder name and relative path), and the date and time that image was taken (as a combined Date/Time Field). Figure 2, top, shows these data fields in the top rows: the greyed out cells are not editable. When the analyst first invokes Timelapse on an image set, Timelapse scans every file for that information and fills in those corresponding data fields.
- Metadata.** Timelapse includes a *metadata viewer*, which the analyst can invoke on one of the images being analyzed and specify what data should be imported. We explain how this works by the example illustrated in Figure 11. The metadata viewer displays all the metadata found in the image as a table. The analyst sees, in the first row of the table in Figure 11, that the camera has recorded some metadata of interest: the 'Ambient Temperature' field that records the temperature at the time the image was taken. As annotated in red in Figure 11, the analyst can link the Ambient Temperature metadata

field to a Timelapse data entry Note field – in this case a field called ‘Temperature’ – simply by selecting both of them. When the analyst clicks the ‘Populate’ button (bottom), the ‘Temperature’ field for each and every image is automatically filled in with the Ambient Temperature metadata value recorded in each image. The process can be repeated for other metadata of interest.

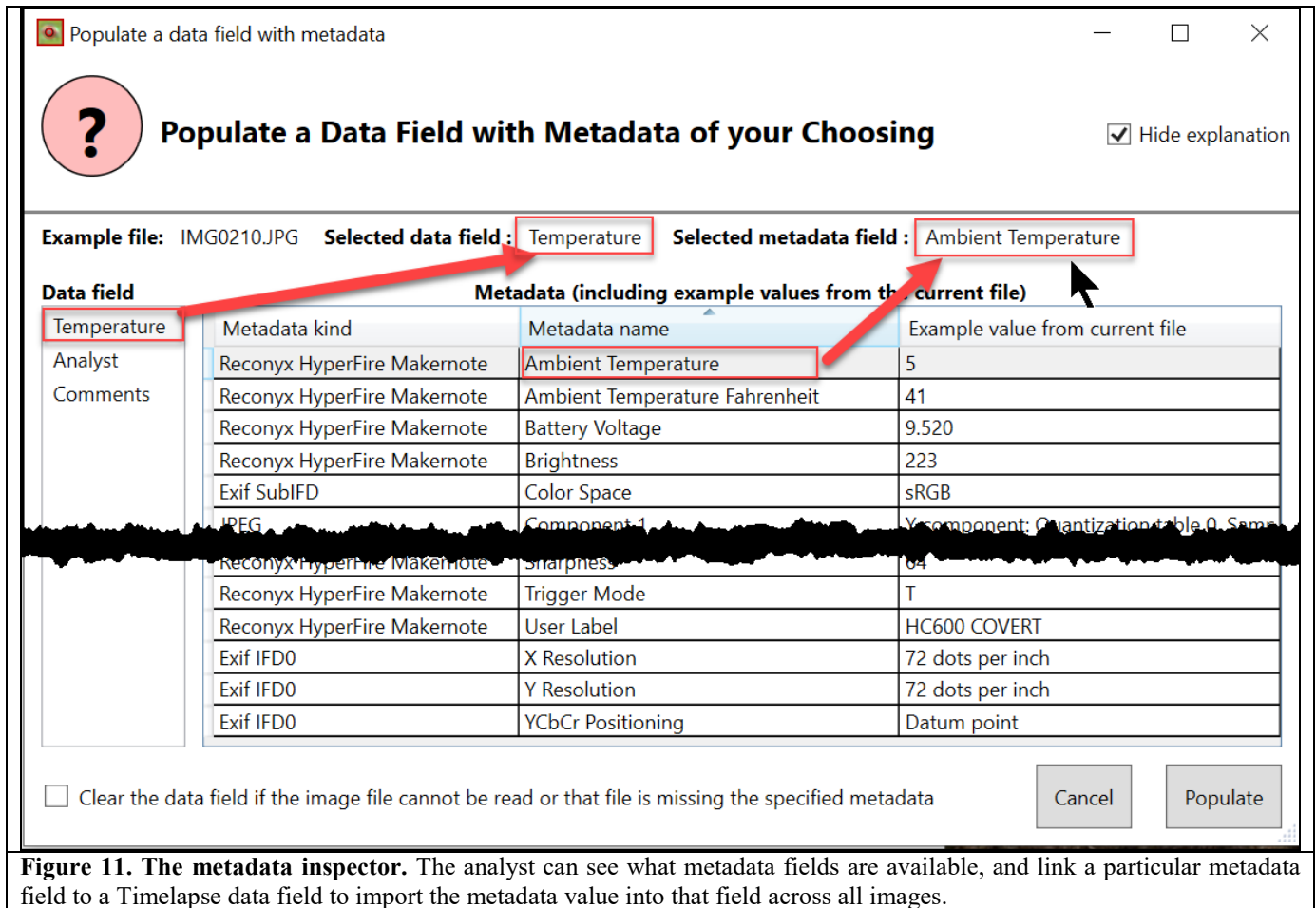


Figure 11. The metadata inspector. The analyst can see what metadata fields are available, and link a particular metadata field to a Timelapse data field to import the metadata value into that field across all images.

Design Pattern. The system should, if plausible, use image analysis techniques to automatically fill in data fields.

Problem – The analyst has to enter information that the computer should be able to recognize by image analysis. Analysts usually have experience using a variety of other image-based systems when doing day to day and recreational tasks. Many include capabilities that recognize aspects of an image, with perhaps face recognition, bar code reading, and text recognition being common examples. Analysts may find it frustrating to enter data that they believe could be detected through image analysis and automatically filled in.

Design approach. Generally speaking, image analysis is the extraction of meaningful data from a digital image. One form of image analysis is image recognition, where complex algorithms use models built upon prior human classification to identify features in an image, such as objects, people, text, faces, and so on. As previously discussed, various researchers are now applying image analysis, and in particular image recognition techniques, to classify images from camera traps. A typical objective is to see how well various recognition algorithms identify animal species (e.g., Norouzzadeha, et al, 2018; Schneider et al, 2018, Tabak et al., 2018; Yousif et al, 2019), and even in recognizing individuals in particular species (e.g., Cheema & Anand, 2017; Crouse et al., 2017). Simpler image analysis methods can also identify other image aspects, e.g., differentiate between color vs. monochrome images, light vs. dark images, and so on. Because image analysis and recognition is not yet full-proof, manual verification of the data will be required, at least for the near future. Thus extracted data should be integrated into the analyst’s workflow in a manner that allows the analysts to check and correct that data as needed.

Timelapse example.

- Dark images.** Some of the agencies we worked with used cameras set in timelapse mode that periodically took images over a day’s 24 hour period. A good number of those images proved of little value because they were too dark (e.g, shots

taken at night-time), and added clutter to the images being reviewed. To help identify overly dark images, Timelapse incorporates an image analyser that automatically classifies images against a user-configurable darkness threshold. Its classification is recorded in the 'Image Quality' data field of every image as either 'Dark' or 'Ok'. Timelapse also includes the ability to filter the displayed images by its data, which we will discuss shortly. Analysts could apply a 'Dark' image filter to display only dark images, where the analyst can quickly review and correct the classification if needed, and perhaps discard those night-time shots. Alternately, the analyst could apply an 'Ok' filter, which displays only the non-dark images.

- b. **Animal detection and recognition.** We are currently working with several vision researchers who specialize in automated object detection (e.g., presence and location of an animal or person) and species recognition. (Microsoft 2019, Schneider et al., 2018). Figure 12 illustrates a Timelapse prototype that imports and displays animal detection data produced by Microsoft's "Megadetector" model (Microsoft 2019). Basically, Megadetector scans all provided images, and outputs data to a file. For each image, Megadetector specifies whether an animal or person is detected, its confidence of correctness, and the coordinates of a bounding box outlining each animal or person's location. Timelapse imports that data, and draws a bounding box atop each identified animal above a detection confidence threshold (set by the analyst). The analyst then uses the standard Timelapse features to eliminate images that don't require further review (e.g., empty images), and to review (and correct if needed) the animal's or person's presence at given confidence levels as needed.

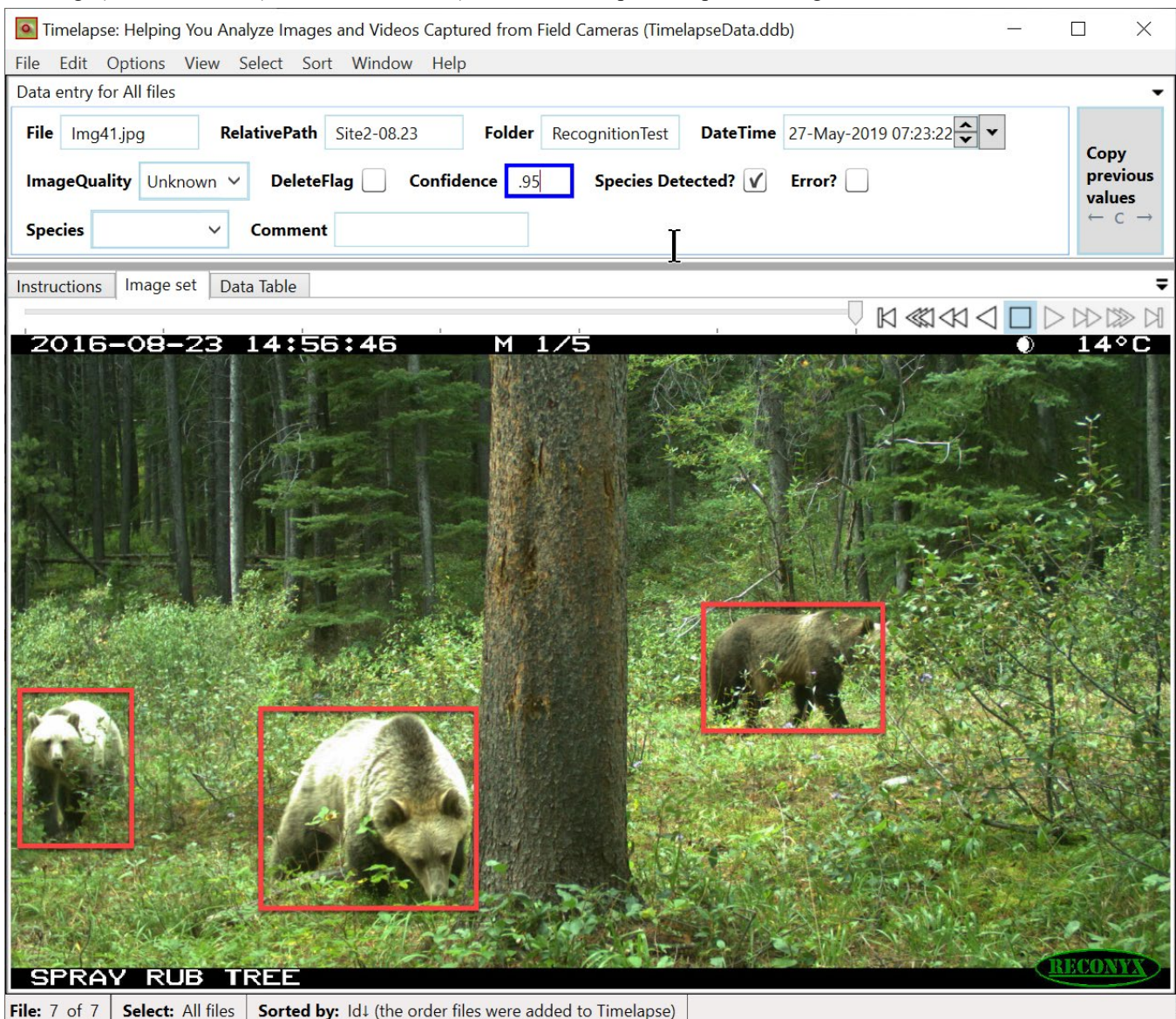


Figure 12. Timelapse prototype incorporating recognition data: bounding boxes are drawn around each suspected species in the image when its detection confidence exceeds a user-defined threshold.

Design pattern: The system should provide facilities to bulk-correct common date/time errors.

Problem – Cameras often record incorrect or ambiguous timestamps. We have observed many issues resulting from the way camera traps record date and time. While the software can automatically import and fill in date/time fields, analysts may have to correct those after the fact. The problem is that it is incredibly time-consuming to manually correct every image's date and time. Common issues we have observed are as follows.

- The camera is not set to the correct date and time when deployed, meaning all date/times are off by a fixed amount.
- The camera does not take into account changes in daylight saving time, which means a large subset of images are off by an hour.
- The camera's internal clock drifts, e.g., it runs slow or fast, which means that the date/time of successive images are increasingly inaccurate.
- The camera records dates ambiguously. For example, consider a date recorded as 02/10/2019. This date can be interpreted as either October 2, 2019 in day / month order, or as February 10 in month / day order. Even worse are cameras that record the year as only the last two digits, for example, 02/10/10 could be interpreted many different ways. This issue is exacerbated by the way different countries set different format standards for encoding dates and times (e.g., see Wikipedia: Date format by country).

Design approach. All the above errors are amenable to bulk-correction, albeit with some manual guidance. For example, if the camera was not set to the correct date and time, the analyst would only have to enter the correct date for the first image. The system could calculate the difference between the two, and then use that difference to time-shift the date and time for all subsequent images. Similarly, the analyst can specify where the daylight savings time change should occur and time-shift previous or subsequent images by plus or minus an hour. To correct for internal clock drifts, the analyst can specify the correct time for the last image, where the system would then calculate a drift factor and adjust the times across all images. When the software detects a possibility for ambiguous dates, it can notify the analyst who can then indicate which date format to apply.

Timelapse example. Because we expect analysts to have to correct dates only infrequently, analysts can raise specialized dialogs for each type of date/time error mentioned above: each dialog includes full instructions and an easy to use interface for specifying how the date should be corrected. For example, Figure 13 illustrates the Timelapse dialog for correcting standard / daylight savings time errors. The analyst navigates to the first image that should be corrected, and then specifies (via various checkboxes) how the correction should be applied. A preview of the corrected date and time is also displayed.

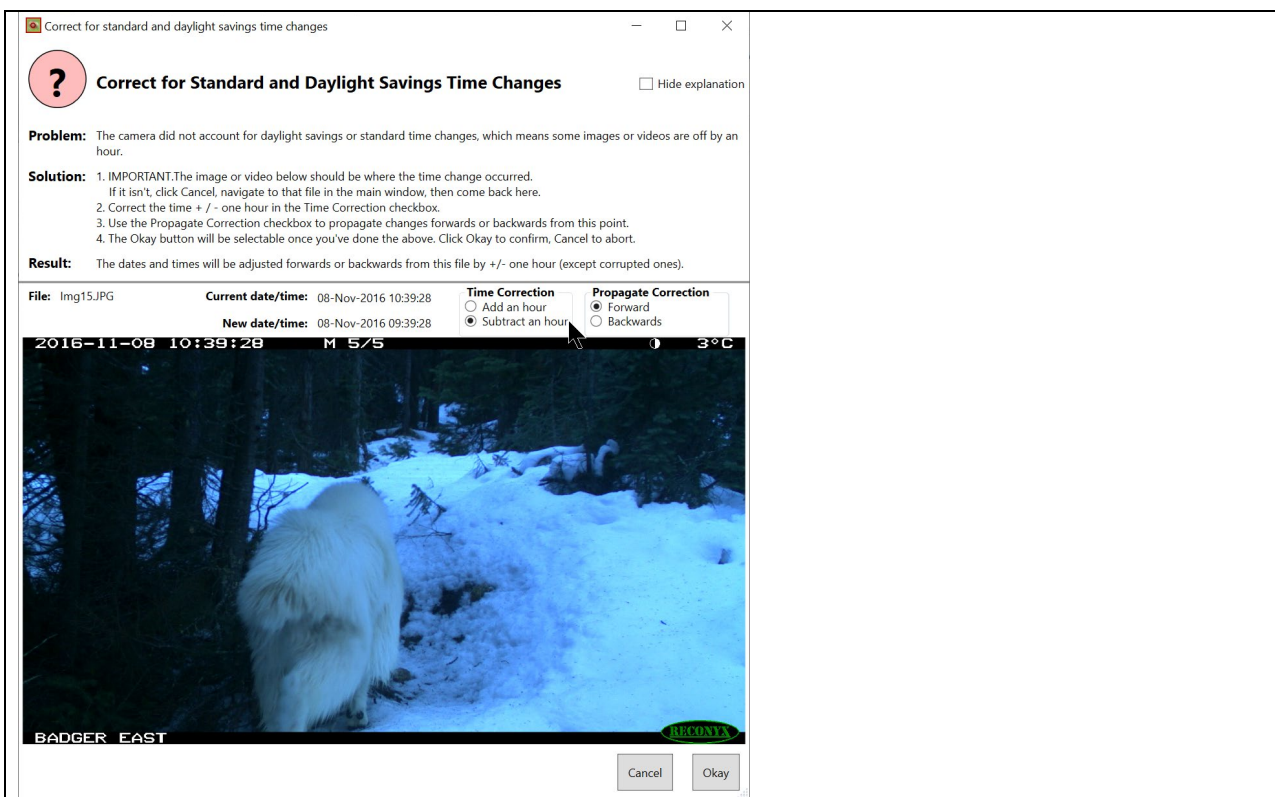


Figure 13. Dialog for correcting daylight savings time

9. THEME: ENTERING REPETITIOUS DATA

Design pattern: It should be easy to re-enter data previously entered elsewhere.

Problem - Similar data is often entered and re-entered over many images. Image sets often comprise subsets of very similar images. For example, a motion-triggered camera may capture a sequence of multiple images of an animal moving through a scene. As another example, an image set can comprise a small set of recurrent but interspersed images, e.g., images containing goats, or elk, or deer, or nothing at all. The data entered that describes these images are often highly similar. Even when the analyst recognizes these similarities, she still has to manually enter the same data per similar image over and over again. This leads to highly repetitious and very time-consuming data re-entry.

Design approach. Various general techniques are known in other domains for re-entering the same data efficiently. Examples include history lists, copying and pasting, predictions based on previous entries, data propagation, and others.

Timelapse example. Timelapse includes several techniques for easing the task of entering repetitive data across multiple images.

a. **Text prediction in a single data field.** As already discussed, Notes include auto-completion capabilities. They maintain a history of previously-typed text entries, and use those to predict the rest of the text as the analyst types.

b. **Propagating data across a single data field.** Every data field includes a pop-up menu that allows the analyst to propagate data across a sequence of images (Figure 14). *Propagate from the last non-empty value to here* uses back-filling. That is, it will copy the last non-empty value entered by the analyst in a data field (e.g., several images back in the sequence) to every intervening empty field up to the current image. For example, the analyst may enter an image's

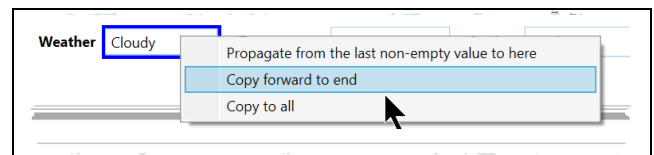


Figure 14. The data field's pop up menu for propagating data

Weather as 'Sunny', then navigate forward through the images until the weather changes, and then backfill the intervening empty fields with that value. *Copy forward to end* is somewhat similar, except it forward-fills the current value to all remaining images in the sequence. It can be re-applied at any time where it over-writes existing values. For example, an analyst may Copy forward Cloudy (as in Figure 14), then move through the sequence until the next non-Cloudy day is noted, enter the new value, and then Copy that forward as well. *Copy to all* copies the current value to all images. For example, the analyst may just enter their name once in the 'Analyst' field and copy that to all images.

c. **Copy Previous Values.** Image sets often contain runs of identical images, where some of the data entered over the next image is identical to what was entered in the previous image. Timelapse supplies a 'Copy Previous Values' button, illustrated in Figure 15.

Pressing this button copies the previous image's values from particular data fields (those set as 'Copyable' in the template: See Figure 2) to the current image's data fields. As illustrated in Figure 15, previews of what fields are affected and the data that will be copied are displayed and highlighted in green when the analyst hovers the mouse over the Copy previous values button.

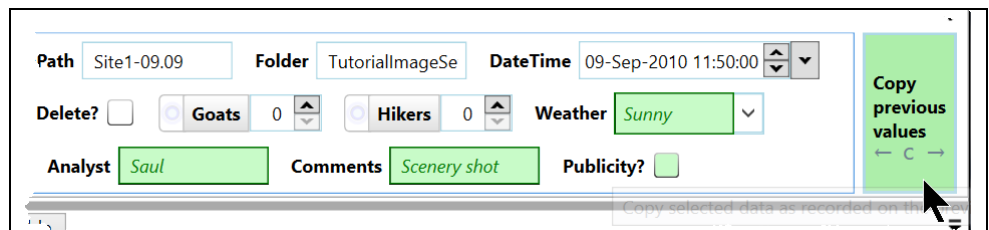


Figure 15. Copy Previous Values button, showing previews of the data to be copied.

d. **Quickpaste: Copying and pasting multiple data fields.** Analysts typically recognize when they entering a small set of similar data patterns over and over again. Timelapse provides QuickPaste as a way for the analyst to capture and name these data entry patterns, where the analyst can then paste that pattern into an image's data field via a single mouse click. Figure 16 below illustrates this through a simple example. The analyst has raised the QuickPaste editor (left) to compose a QuickPaste entry: she has titled the entry 'No goats, sunny', and has selected and filled in which data fields should be used (Goats, Weather, Analyst, Comments, Publicity) and the values to be pasted. This entry is then added to the list of other QuickPaste entries in the QuickPaste window (right side). The analyst can then use the QuickPaste window to select and paste particular patterns into the image's data entry fields. As illustrated in Figure 16, when she hovers over an entry, a preview of the values to be pasted appear in the affected data fields (highlighted in green). Clicking the entry pastes those values into the field. While requiring some initial setup to create these custom entries, QuickPaste becomes a very effective and efficient way for entering common data patterns.

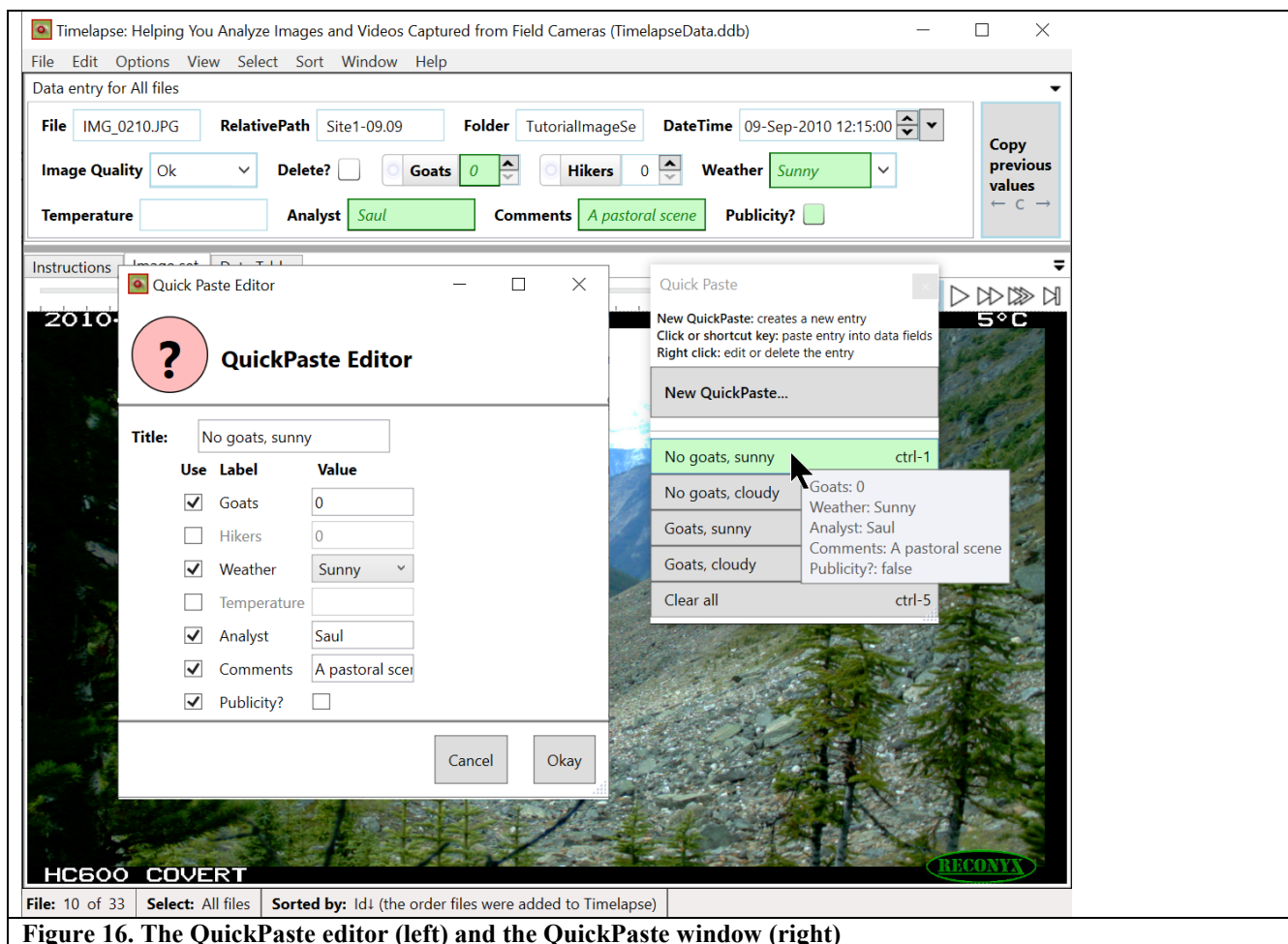


Figure 16. The QuickPaste editor (left) and the QuickPaste window (right)

Design pattern: Allow the analyst to inspect and bulk-enter data for multiple images at a time.

Problem - Reviewing and entering repetitive data image by image can be inefficient. Most image packages display a single image at a time, where the analyst has to inspect and enter data for them individually.

Design approach. The system should provide facilities for displaying multiple images at a time (e.g., a table of large thumbnails). The analyst should be able to select particular images with common features, and then bulk-entering data for those selected images all at once. The analyst should also be able to choose the appropriate thumbnail size, as the features of interest needs to be discernable.

Timelapse example. The overview supplied in Timelapse, discussed above and previously illustrated in Figure 7, allows the analyst to review multiple images at the same time. The analyst can quickly trade-off the number of images displayed vs. the image size (to optimize just-discernable features with the number of images shown) by zooming in or out of different overview levels with the scroll wheel. The analyst can then select and bulk-edit data for one or more of those images. For example, Figure 7 shows how the analyst selected only those images with a full view of a goat in it (the first 5 images), where she has entered a '1' in the Goats Counter field and 'Full body view' in the Comment field. Those values are then applied to all the selected images. Interface subtleties are also addressed. As multiple selections are done, the data fields and their contents are adjusted to reflect that selection. For example, and as also shown in Figure 7, the DateTime data field is disabled as bulk-editing that field makes little sense. If a data field in the selected images all share the same data value, that value is displayed. Otherwise a '...' symbol is displayed to indicate that their values differ.

10. THEME: SORTING AND FILTERING THE IMAGE SEQUENCE

Design pattern: Allow the analyst to sort images by one or more criteria.

Problem – Images are often presented in a single sort order, usually based on their file name, which may not reflect how the analyst wants to view them. Analysts usually inspect images as a sequence, one after the other. Thus the way images are

ordered (sorted) can affect what they see and how they interpret images as events unfolding over time. Consider the example of a motion-triggered camera taking images of one or more animals moving through the scene. If the presentation sequence is in time-order, the analyst will recognize that those images relate to one another, as they are capturing a single event. As another example, the analyst may wish to review already classified images ordered by a combination of criteria. For example, the analyst may want to get a sense of whether the number of goats using the pasture in Figure 3 is correlated to weather conditions. This can be done by ordering images by weather and then by the number of goats. The problem is that most systems typically order images only by its file name, and do not allow any other sorting capabilities.

Design approach. Providing the ability to sort by date/time rather than file name is perhaps the most fundamental sort capability that should be included. While cameras typically add a sequence number to a file name as images files are created, there is no guarantee that they will be presented in time order e.g., alphabetically sorted files named 1.jpg, 2.jpg... 10.jpg would be presented as 1.jpg, 10.jpg, 2.jpg..., which breaks time ordering. Ideally, the software will also allow the analyst to sort on any data field or combination of fields and their data values.

Timelapse example. Timelapse provides a sorting capability based on one or two data fields of the analyst's choosing. The analyst can quickly select (via a menu) common sorting criteria including image load order, date/time order, how images are organized into folders, and by particular data entry field contents. The analyst can also raise a custom sort dialog (Figure 17 below), where she can choose a primary and secondary sorting criteria from a drop-down menu that lists labels for the data fields. In this case, she is sorting by Weather and then by Goats. Images are then presented in that sort order. The rows in the data table in Figure 8 are also updated to that sort order.

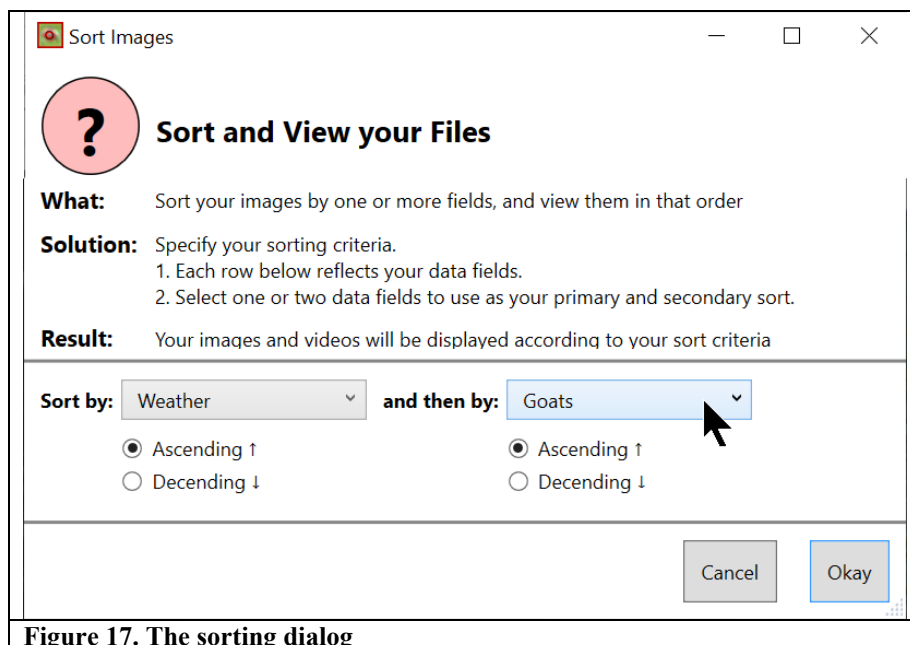


Figure 17. The sorting dialog

Design pattern: Allow the analyst to specify criteria that filters which images are displayed.

Problem - The analyst may need to view a particular subset of images. Analysts may, at times, be interested in only a subset of the available files. Yet finding and viewing the images in this subset can be problematic, especially with large image sets comprising tens of thousands of files. As one example, the analyst may want to verify and possibly correct prior image classification category, e.g., that all system-classified dark images are indeed dark, that images classified by another analyst as 'Goats' all contain goats, and so on. As another example, the analyst may be interested in only those files taken at a certain site and between particular dates. As yet another example, the analyst may want to review a particular image classification in order to choose an archetypical image, for example, an excellent image of a goat to be used for publicity purposes.

Design approach. The system should provide the analyst with a query facility and search engine. The analyst should be able to specify a search query, where the system filters images so that it only displays images matching that query. Query criteria should include queries against the values recorded in the image data fields.

Timelapse example. Timelapse incorporates a free database (SQLite: <http://www.sqlite.org>) to store the data entered by the analyst. SQLite includes a query language for searching for matching records. Thus Timelapse can perform any standard

database search against that data, where search results are returned as records describing the matching images. Those images are then displayed. However, it is unrealistic to expect analysts to compose cryptic SQL query expressions. As a better alternative, Timelapse displays a dialog box listing all data fields, as illustrated in Figure 18. The analyst then composes a query by selecting the data fields of interest, and then specifying the values that should be matched. The system translates that into an SQL query, and returns only those images that match the query. For example, in Figure 18, the analyst is interested in the interaction between goats and hikers and wishes to see only those images that have both a goat and a hiker in it. The analyst selects the Goats and Hikers data fields for use (the ‘Select’ column on the left), and has specified that both have values greater than 0 (the ‘Expression’ column). The ‘AND’ checkbox at the top indicates that both those constraints must be satisfied. Feedback (bottom right) indicates that 3 files match that query. After clicking Okay, only those three images will be available for navigation and display. Had the analyst had clicked the ‘OR’ checkbox instead, then all returned images would contain either one or more goats, or one or more hikers, or both. The Timelapse Sort function can also be applied to the results, for example, to show all images with both goats and hikers, but sorted by the number of goats and then by the number of hikers.

Design pattern: The system should identify and group episodes of time-related images.

Problem - The analyst may need to consider images taken over a short time period as a unit. As previously mentioned, camera traps set in motion capture mode are often triggered when an animal or herd is moving through a scene. This can result in a burst of images that capture that activity, which we define as an *episode*. Episodes are sometimes treated differently than individual images. For example, we saw analysts manually determine which images fall into an episode (e.g., by examining their time-stamp), count the unique wildlife seen in that episode, and enter that data into only a single image. They do this to avoid inflating the number of wildlife present. To illustrate, consider the analyst who has to count the number of hikers using a trail. A single hiker may appear on several images over time, perhaps due to motion triggering, or because the hiker is milling about in the camera’s field of view. To avoid double-counting, the analyst would only count the hiker once in this series. The problem is that it is laborious for the analyst to recognize which images belong together in an episode.

Design approach. Various strategies can be used to identify episodes. For example, some cameras include metadata that indicate whether an image is part of a motion-capture sequence, as well its position in that sequence (e.g., 1/5, 2/5, etc.). While useful, it is limited as an episode can easily comprise two or more back to back motion-capture sequences. Alternately, a reasonable heuristic is to have the system examine the time interval between time-ordered images. If the interval is small, the system would group them together as part of an episode.

Timelapse example. Timelapse uses the heuristic above, where the analyst can ask it to group together images separated by a small user-configurable time interval. Timelapse then annotates each image to indicate how images relate to one another as an episode. Figure 19 is similar to Figure 7, except that it now illustrates how episode annotations appear in the overview. The first image in an episode is colored red (top left) so that the analyst can visually identify the start of the episode. That and subsequent images in the episode are given a sequence number (e.g., 1/3, 2/3, 3/3). A timestamp is also overlaid atop the image, so that the analyst can examine the time differences between those images if needed. If an image does not belong to an episode, it is marked as ‘Single’ (not shown). In Figure 19, the first ten and the last five images are identified as two different episodes of a goat walking through the scene. In this case, the analyst does not want to double count the same goat. Consequently, she selects the best image in each episode (Img04 and Img15), and increments the Goats counter of only that image.

Select and view a subset of your files

Select and View a Subset of your Files ☒ Hide explanation

☒ And to select images and videos matching all selected search terms
☐ Or to select images and videos matching any one of the selected search terms
(Multiple DateTime selections are always And'ed together)

Select	Label	Expression	Value	Search Query
<input type="checkbox"/>	File	=		
<input type="checkbox"/>	RelativePath	=		
<input type="checkbox"/>	DateTime	≥	15-Aug-2010 18:15:00	
<input type="checkbox"/>	DateTime	≤	15-Aug-2010 18:15:00	
<input type="checkbox"/>	Image Quality	=		
<input type="checkbox"/>	Delete?	=		
<input checked="" type="checkbox"/>	Goats	>	0	GoatCnt > 0 And
<input checked="" type="checkbox"/>	Hikers	>	0	HikerCnt > 0
<input type="checkbox"/>	Weather	=		
<input type="checkbox"/>	Temperature	=		
<input type="checkbox"/>	Analyst	=		
<input type="checkbox"/>	Comments	=		
<input type="checkbox"/>	Publicity?	=		

Reset to All Images 3 files match your query Cancel Okay

Figure 18. The query dialog for filtering images from view.

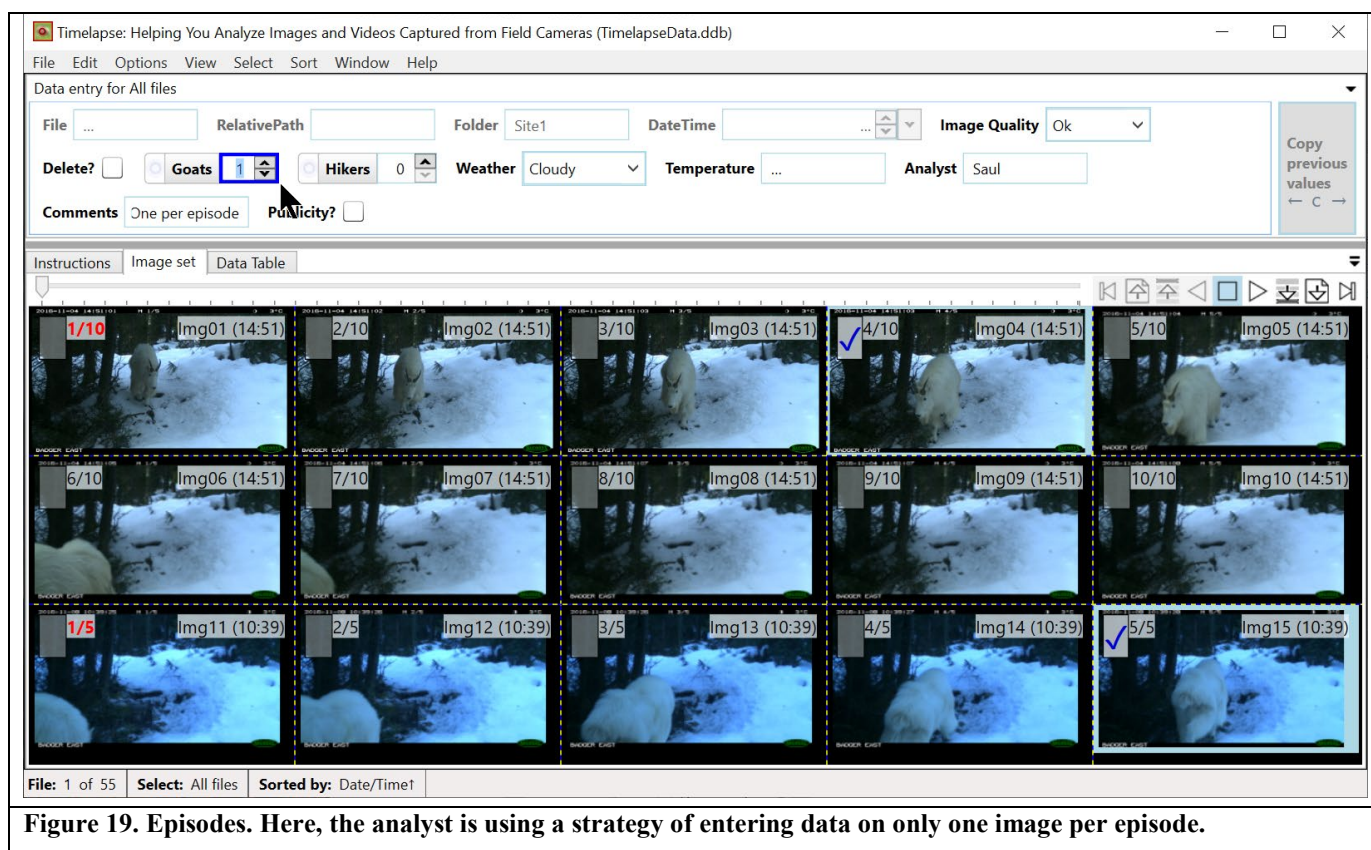


Figure 19. Episodes. Here, the analyst is using a strategy of entering data on only one image per episode.

11. DISCUSSION

Decisions on what software is used to inspect and encode image data has consequences on how well an analyst can perform their job. Yet we question how some agencies make their decision. We have seen some consider only the stock software available on typical computers: e.g., Microsoft Photo Viewer to view images, and an Excel spreadsheet for data entry. This is inefficient. For example, we studied how analysts entered data using spreadsheets vs. an earlier version of Timelapse. Timelapse provided time improvements of ~200% or more, which translates into significant cost savings. We saw other agencies use either researcher-based software or the stock software that came bundled with their cameras without considering the consequences of that choice on the analyst. Some agencies may also make their choice based on other factors, such as how the software stores data in a format amenable to standardization or later analysis vs. how that data is actually entered by analysts. We advocate that decisions on which software is used should deeply consider how well they support the analysts' tasks. The design patterns described earlier should be part of that consideration. Poor system choices imply tedious data entry, are error-prone (which affects the validity of the collected data), are morale-sucking, and – in the long run – are very expensive in terms of analyst time.

While our design patterns identify various problems faced by analysts, we recognize that they range in seriousness, in frequency of occurrence, in applicability to particular projects, and in consequences if they are not addressed. We also recognize that our catalog of design patterns is just a starting point and future work is required: there are surely other problems and design patterns that could and should be articulated and considered in camera trap analysis design. For example, if image analysis is done through crowdsourcing and citizen science (Swanson et al., 2016), design patterns specific to that audience would likely emerge. Design patterns can also extend beyond interface features. For example, they can recognize and address the problems related to data management issues (e.g., Ivan and Newkirk, 2016), data validation, and data standardization and scaling across the field (e.g., Steenweg et al, 2017).

We stress that design patterns are not 'feature list' of specific solutions. Rather, each design pattern suggests a design approach - rather than a definitive solution - that can be adapted, refined, and specialized to best fit the project, the background and needs of the analysts, and the equipment available. Each design pattern can also inform decision-making. If the problem and solution is relevant, that should become a factor influencing the requirements analysis of the software being developed or for a manager deciding between available software systems.

We also show how our own Timelapse system implements the design pattern. These are intended to serve as concrete examples rather than prescriptions. Of course, the specific techniques used by Timelapse could be implemented as is in other camera trap systems. However, we recognize – and indeed encourage – future system designers to see beyond our own solutions, where they should seek solutions that implement the design pattern in even better ways. For example, Timelapse was intentionally designed to work on lowest common denominator computers typically available to analysts: Microsoft Windows running a keyboard and mouse on a conventional low-cost computer as found in many agencies. Thus its design eschewed more modern interaction techniques, such as touch interaction, as we felt it would limit its deployment. If a system such as Timelapse were redesigned to run on (say) a touch-based tablet, we would expect different design solutions that still follow the above design pattern recommendations. Similarly, Timelapse was designed to work off-line so analysts could work in the field on disconnected laptops. If Timelapse were redesigned to work as a networked client or over the web, design solutions would have to account for performance aspects such as network bandwidth and latency that could affect responsiveness and rapid image display.

As mentioned, we recognize that our list of design patterns are incomplete, where future work should elicit other design patterns to produce a comprehensive catalog. Researchers should continually conduct interviews and observation of analysts as they work to gain an even more nuanced understanding of their core tasks and problems. Because camera traps are broadly used for many quite different purposes, domain-specific design patterns should be developed. Other software systems should be reviewed and compared for how they address problems and deliver solutions not covered by Timelapse, and whether those can be encapsulated as useful design patterns. As well, our design patterns are limited to only the analyst's interface for inspecting images and entering data. Future work should consider design patterns for other related tasks. One example concerns interface patterns that suggest how a project manager can view and manage data within and across projects. To illustrate, the Reconyx MapView software (Reconyx, 2016) includes a map interface that lets the project manager or analyst geo-locate study sites and stations onto it, and which lets them drill down into the captured data.

Finally, we recognize that elements of various design patterns are based on aspects well-known within the field of Human-Computer Interaction, Information Visualization, and Experience Design. These fields have a rich literature of research, practitioner's guides, and texts relating to the design of systems for human use, including methodologies that describe how to test how well a person can use that system and its features (e.g., Shneiderman et al., 2016; Spence, 2014). As well, various stock components and interaction techniques are readily available in software development tools, where most are based upon best practices of user interaction. The catch is that decisions of what is relevant must still be made on the needs of the domain being considered. This is the purpose of this paper, where it identifies problems and solutions as design patterns relevant to the domain of camera trap image analysis.

Author contributions. See the methodology section, which describes the various roles played by the authors in more detail. Greenberg developed the Timelapse software and was the primary author of the design patterns listed here. Godin and Whittington contributed regularly to the Timelapse software design via on-going discussions of its features and its use by their team of analysts. They also reviewed and contributed to various drafts of this paper.

Timelapse accessibility. Timelapse is an open source project written in C#/WPF and available on the Github repository: <https://github.com/saulgreenberg/Timelapse>. For even easier access, Timelapse software, installation instructions, tutorial documentation (including example image and template files), and mailing list information are freely available at <http://saoul.cpsc.ucalgary.ca/timelapse>. Project managers and analysts are invited to download Timelapse, and developers are invited to modify or enhance the software as needed. Finally, Timelapse is actively maintained and supported by the first author of this paper. Contact saoul@ucalgary.ca for more information.

Acknowledgements. This research was partially funded by the National Science and Engineering Research Council. We interacted with a large number of Timelapse users over the years. We listened to their needs, their feature requests, heard their feedback of using Timelapse, and watched them use Timelapse on their own images. That information was invaluable for designing the successive improved versions of the system over time. We also thank Todd West for his contributions to earlier versions of Timelapse. Images are courtesy of Parks Canada.

REFERENCES

1. Ahumada J.A., Hurtado J., Lizcano D. (2013) Monitoring the Status and Trends of Tropical Forest Terrestrial Vertebrate Communities from Camera Trap Data: A Tool for Conservation. PLoS ONE 8(9): e73707. <https://doi.org/10.1371/journal.pone.0073707>
2. Alexander, C. (1977) A Pattern Language: Towns, Buildings, Construction. Oxford University Press.

3. Blake, J.G., Mosquera, D., Loisel, B.A., Romo, D. and Swing. K. (2017) Effects of human traffic on use of trails by mammals in lowland forest of eastern Ecuador, *Neotropical Biodiversity*, 3:1, 57-64, <https://doi.org/10.1080/23766808.2017.1292756>
4. Borchers, J. (2001) A Pattern Approach to Interaction Design. *AI & Society*, 12:359-376, Springer.
5. Bubnicki, J.W., Churski, M. and Kuijper, D.P.J (2016) Trapper: An open source web-based application to manage camera trapping projects. *Methods in Ecology and Evolution*, 7(10), 1209-1216. <https://doi.org/10.1111/2041-210x.12571>
6. Burton, C.A., Neilson, E., Moreira, D., Ladle, A., Steenwag, R., Fisher, J.T., Bayne, E. and Boutin, S. (2015) Wildlife camera trapping: A review and recommendations for linking surveys to ecological processes. *J. Applied Ecology*, 52:675-685. <https://doi.org/10.1111/1365-2664.12432>
7. Campbell, J.M. (2010) Seeing is Believing: Using Digital Cameras to Monitor Trail use in Riding Mountain National Park. S. In Bondrup-Nielsen, K. Beazley, G. Bissix, D. Colville, S. Flemming, T. Herman, M. McPherson, S. Mockford, and S. O'Grady (Eds). *Ecosystem Based Management: Beyond Boundaries. Proceedings of the Sixth International Conference of Science and the Management of Protected Areas*.
8. Carpendale, S., Light, J. and Pattison, E. (2004) Achieving higher magnification in context. In *Proceedings of the 17th annual ACM symposium on User interface software and technology (UIST '04)*. ACM, New York, NY, USA, 71-80. DOI: <https://doi.org/10.1145/1029632.1029645>
9. Cheema, G.S. and Anand, S. (2017) Automatic Detection and Recognition of Individuals in Patterned Species. In: Altun Y. et al. (eds) *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2017. Lecture Notes in Computer Science*, vol 10536. Springer, Cham https://link.springer.com/chapter/10.1007/978-3-319-71273-4_3
10. Clevenger, A.P. and Sawaya, M.A. (2010) Piloting a Non-Invasive Genetic Sampling Method for Evaluating Population-Level Benefits of Wildlife Crossing Structures. *Ecology and Society*, Vol. 15, No. 1 (Mar). <http://www.ecologyandsociety.org/vol15/iss1/art7/>
11. Crouse, D., Jacobs, R.L., Richardson, Z., Klum, S., Jain, A. Baden, A.L. and Teco, S.R. (2017) LemurFaceID: a face recognition system to facilitate individual identification of lemurs. *BioMed Central Zoology*, 2:2. <https://doi.org/10.1186/s40850-016-0011-9>
12. Fairfax, R.J., MacKenzie Dowling, R. Neldner, V.J. (2014) The use of infrared sensors and digital cameras for documenting visitor use patterns: a case study from D'Aguilar National Park, south-east Queensland, Australia. *Journal of Current Issues in Tourism*, Volume 17, Issue 1. <https://doi.org/10.1080/13683500.2012.714749>
13. Fisher, J.T. and Burton, A.C. (2018) Wildlife winners and losers in an oil sands landscape. *The Ecological Survey of America*, 16(5): 323-328, <https://doi.org/10.1002/fee.1807>
14. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson.
15. García-Salgado G, Rebollo S, PérezCamacho L, Martínez-Hestekamp S, Navarro A, Fernández-Pereira J-M (2015) Evaluation of TrailCameras for Analyzing the Diet of Nesting Raptors Using the Northern Goshawk as a Model. *PLoS ONE*, 10(5): <https://doi.org/10.1371/journal.pone.0127585>, May.
16. Glover-Kapfer, P., Soto-Navarro, C.A. and Wearn, O.R. (2019) Camera-trapping version 3.0: Current constraints and future priorities for development. *Remote Sensing in Ecology and Conservation*, <https://doi.org/10.1002/rse2.106>
17. Goswami, V.R., Madhusudan, M.D., Ullas Karanth, K. (2007) Application of photographic capture-recapture modelling to estimate demographic parameters for male Asian elephants. *Animal Conservation*, 10, 391-399. <https://doi.org/10.1111/j.1469-1795.2007.00124.x>
18. Greenberg, S. (2019) The Timelapse User Guide version 2.2.2.4. Updated versions of this manual available at <http://saul.cpsc.ucalgary.ca/timelapse/pmwiki.php?n=Main.UserGuide>
19. Greenberg, S. and Godin, T. (2015) A Tool Supporting the Extraction of Angling Effort Data from Remote Camera Image (Feature Article). *Fisheries Magazine*, 40(6):276-287. American Fisheries Society, June. <https://doi.org/10.1080/03632415.2015.1038380>
20. Heilbrun, R. D., Silvy, N. J., Peterson, M. J. and Tewes, M. E. (2006), Estimating Bobcat Abundance Using Automatically Triggered Cameras. *Wildlife Society Bulletin*, 34: 69-73. [http://doi.org/10.2193/0091-7648\(2006\)34\(69:EBAUAT\)2.0.CO;2](http://doi.org/10.2193/0091-7648(2006)34(69:EBAUAT)2.0.CO;2)

21. Hossain, A.N.M., Barlow, A, Greenwood Barlow, C., Lynam, AJ, Chakma, S., Savini, T. (2016) Assessing the efficacy of camera trapping as a tool for increasing detection rates of wildlife crime in tropical protected areas, *Biological Conservation*, Volume 201, Pages 314-319, ISSN 0006-3207, <https://doi.org/10.1016/j.biocon.2016.07.023>
22. Ivan, J.S. and Newkirk, E.S. (2016) Cpw Photo Warehouse: a custom database to facilitate archiving, identifying, summarizing and managing photo data collected from camera traps. *Methods in Ecology and Evolution*, 7(4):499-504, <https://doi.org/10.1111/2041-210X.12503>
23. Jumeau, J., Petrod, L. and Handrich, Y. (2017) A Comparison of Camera Trap and Permanent Recording Video Camera Efficiency in Wildlife Underpasses. *Ecology and Evolution*, Volume 7, Issue 18, pages 7399-7407, September. <https://doi.org/10.1002/ece3.3149>
24. Karanth, K.U. and Nichols, J.D. (1998) Estimation of tiger densities in India using photographic captures and recaptures. *Ecology* 79:2852-2862. [https://doi.org/10.1890/0012-9658\(1998\)079\(2852:EOTDII\)2.0.CO;2](https://doi.org/10.1890/0012-9658(1998)079(2852:EOTDII)2.0.CO;2)
25. Karanth, K.U., Nichols, J.D., Samba Kumar, N. and Hines, J.E. (2006) Assessing tiger population dynamics using photographic capture-recapture sampling. *Ecology* 87(11). [https://doi.org/10.1890/0012-9658\(2006\)87\(2925:ATPDUP\)2.0.CO;2](https://doi.org/10.1890/0012-9658(2006)87(2925:ATPDUP)2.0.CO;2)
26. Krishnappa, Y.S. and Turner, W.C. (2014) Software for minimalistic data management in large camera trap studies. *Ecological Information*, (24):11-16. <https://doi.org/10.1016/j.ecoinf.2014.06.004>
27. Laskin, D.N., McDermid, G. J., Nielsen, S. E., Marshall, S. J., Roberts, D. R. and Montaghi, A. (2019) Advances in phenology are conserved across scale in present and future climates. *Nature Climate Change* 9:419-425. <https://doi.org/10.1038/s41558-019-0454-4>
28. Microsoft (2019). AI for Earth Camera Trap Image Processing API. Github repository of its Megadetector recognizer, <https://github.com/Microsoft/CameraTraps> . Accessed May, 2019.
29. Mills, L. S., Bragina, E. V., Kumar, A. V., Zimova, M., Lafferty, D. J. R., Feltner, J., Davis, B. M., Hackländer, K., Alves, P. C., Good, J. M., Melo-Ferreira, J., Dietz, A., Abramov, A. V., Lopatina, N. and Fay. K. (2018). Winter color polymorphisms identify global hot spots for evolutionary rescue from climate change. *Science*, 359(6379), pp. 1033-1066. <http://doi.org/10.1126/science.aan8097>
30. Nielsen, J. (1993) *Usability Engineering*, Academic Press.
31. Norouzzadeha, M.S., Nguyenb, A., Kosmalac, M., Swanson, A. Palmere, M.S., Packere, C. and Clunea, J. (2018) Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *PNAS (Proc. National Academy of Sciences of the USA)* , 115 (25), E5716-E5725, <https://doi.org/10.1073/pnas.1719367115>
32. Oberoslerab, V., Groff, C., Lemma, A. Pedrini, P., Rovero, F. (2017) The influence of human disturbance on occupancy and activity patterns of mammals in the Italian Alps from systematic camera trapping. *Mammalian Biology*, Volume 87, Pages 50-61, November. Elsevier. <https://doi.org/10.1016/j.mambio.2017.05.005>
33. O'Brien, T.G. and Kinnaird, M.F. (2010) Estimation of Species Richness of Large Vertebrates Using Camera Traps: An Example from an Indonesian Rainforest. In O'Connell, A.F., Nichols, J.D., Karanth, K. (Eds.) *Camera Traps in Animal Ecology: Methods and Analysis*, Springer. ISBN 978-4-431-99495-4. <http://www.springer.com/us/book/9784431994947>
34. Pollock, S. Z., Nielsen, S. E. and St. Clair, C. C. 2017. A railway increases the abundance and accelerates the phenology of bear-attracting plants in a forested, mountain park. *Ecosphere* 8(10):e01985. <https://doi.org/10.1002/ecs2.1985>
35. Reconyx, Inc. (2016) Reconyx MapView Professional Software (2016) Version 3.7.2.2, including user guide. <https://www.reconyx.com/software/mapview>.
36. Rollack, C., Wiebe, K, Stoffel, M.J. and Houston, S. (2013) Turkey Vulture Breeding Behavior Studied with Trail Cameras. *Journal of Raptor Research*, 47(2):153-160 (2013). <https://doi.org/10.3356/JRR-12-40.1>
37. Rowcliff, J.M. (2008) Surveys using camera traps: are we looking to a brighter future? *Animal Conservation*, 11(3). <https://doi.org/10.1111/j.1469-1795.2008.00180.x>
38. Royle, J. A., A. K. Fuller, and C. Sutherland. 2018. Unifying population and landscape ecology with spatial capture-recapture. *Ecography* 41:444–456. <https://doi.org/10.1111/ecog.03170>

39. Schneider, S., Taylor, G.W. and Kramer, S.C. (2018) Deep learning object detection methods for ecological camera trap data. Proc. IEEE 15th Conference on Computer and Robot Vision (May 8-10, Toronto, Ontario, Canada), pp. 321-328. <https://doi.org/10.1109/CRV.2018.00052>
40. Scotson, L. , Johnston, L. R., Iannarilli, F. , Wearn, O. R., Mohd-Azlan, J. , Wong, W. , Gray, T. N., Dinata, Y. , Suzuki, A. , Willard, C. E., Frechette, J. , Loken, B. , Steinmetz, R. , Moßbrucker, A. M., Clements, G. R. and Fieberg, J. (2017) Best practices and software for the management and sharing of camera trap data for small and large scales studies. Remote Sensing in Ecology and Conservation, 3: 158-172. <https://doi.org/10.1002/rse2.54>
41. Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N. and Diakopoulos, N. (2016) Designing the User Interface: Strategies for Effective Human-Computer Interaction (6th Edition). Pearson.
42. Spence, R. (2014) Information Visualization: An Introduction, 3rd Edition, Springer.
43. Steenweg, R., Hebblewhite, M., Kays, R., Ahumada, J., Fisher, J.T., Burton, C., Townsend, S.E., Carbone, C., Rowcliffe, J.M., Whittington, J., Brodie, J., Royle, J.A., Switalski, A., Clevenger, A.P., Heim, N. and Rich, L.N. (2017) Scaling-up camera traps: monitoring the planet's biodiversity with networks of remote sensors. Frontiers in Ecology and the Environment, 15(1):26-34. <https://doi.org/10.1002/fee.1448> .
44. Swann, D.E., Kawanishi, K. and Palmer, J. (2010) Evaluating Types and Features of Camera Traps in Ecological Studies: A Guide for Researchers. In O'Connell, A.F., Nichols, J.D., Karanth, K. (Eds.) Camera Traps in Animal Ecology: Methods and Analysis, Springer. ISBN 978-4-431-99495-4. <http://www.springer.com/us/book/9784431994947> .
45. Swanson, A., Kosmala, M., Lintott, C. and Packer, C. (2016) A generalized approach for producing, quantifying, and validating citizen science data from wildlife images. Conservation Biology Volume 30, Issue 3, Wiley. <https://doi.org/10.1111/cobi.12695>
46. Swanson, A., Kosmala, M., Lintott, C., Simpson, R. Smith, A. Packer, C. (2015) Snapshot Serengeti, High Frequency Annotated Camera Trap Images of 40 Mammalian Species in an African Savanna. Scientific Data 2, Article number: 150026 (nature.com), <https://doi.org/10.1038/sdata.2015.26>
47. Tabak, M.A., Norouzzadeh, M.S., Wolfson, D.W., Sweeney, S.J., Vercauteren, K.C., Snow, N.P., Haseeth, J.M., Di Salvo, P.A., Lewis, J.S., White, M.D., Teton, B., Beasley, J.C., Schlichting, P.E., Boughton, R.K., Wight, B., Newkirk E.S., Ivan, J.S., Odell, E.A., Brook, R.K., Lukacs, P.M., Moeller, A., Mandeville, E.G., Clune, J. and Miller, R.S. (2018) Machine learning to classify animal species in camera trap images: Applications in Ecology. *Methods in Ecology and Evolution*, 2018:1-6, British Ecological Society. <https://doi.org/10.1111/2041-210X.13120>
48. Tobler M.W, Zúñiga Hartley A, Carrillo-Percastegui S.E. and Powell G.V. (2015) Spatiotemporal hierarchical modelling of species richness and occupancy using camera trap data. J Applied Ecology 52: 413–421. <http://doi.org/10.1111/1365-2664.12399>
49. Ware, C. and Lewis, M. (1995). The DragMag Image Magnifier. Chi '95 Video Program. From ACM Conference on Human Factors in Computing System, May 7-11, Denver. ACM Press. Videotape.
50. Wearn, O.R. and Glover-Kapfer, P.G. (2017) Camera-trapping for conservation: A guide to best-practices. WWF Conservation Technology Series 1(1), WWF-UK, Woking, United Kingdom. <https://www.wwf.org.uk/conservationtechnology/documents/CameraTraps-WWF-guidelines.pdf>
51. Whittington, J., Low, P. and Hunt, B. (2019) Temporal road closures improve habitat quality for wildlife. Nature: Scientific Reports, 9:3772. Springer. <https://doi.org/10.1038/s41598-019-40581-y>
52. Whittington, J., M. Hebblewhite, and R. B. Chandler. 2018. Generalized spatial mark-resight models with an application to grizzly bears. Journal of Applied Ecology 55:157-168.
53. WildTrax (2019). Website of the Alberta Biodiversity Monitoring Institute, Biological Sciences Centre, University of Alberta, Edmonton, Canada. Last accessed April 7, 2019. <http://www.wildtrax.ca/home.html>
54. Young, S., Rode-Margono, J. and Amin, R. (2018) Software to facilitate and streamline camera trap data management: A review. Ecology and Evolution, 8(19), <https://doi.org/10.1002/ece3.4464>
55. Yousif, H., Yuan, J., Kays, R. and He, Z (2019). Animal Scanner: Software for classifying humans, animals, and empty frames in camera trap images. Ecology and Evolution, 9, 1578–1589, Wiley. <https://doi.org/10.1002/ece3.4747>