Instituto Universitário de Lisboa

Department of Information Science and Technology

Application for Photogrammetry of Organisms

Francisco Faria Aleixo

Dissertation submitted as partial fulfillment of the requirements for the degree of

Master in Computer Engineering

Supervisor:
PhD Luís Eduardo de Pinho Ducla Soares, Assistant Professor,
ISCTE-IUL

Co-supervisor:
PhD Paulo Jorge Lourenço Nunes, Assistant Professor,
ISCTE-IUL

External co-supervisor:
PhD Rui Conde de Araújo Brito Prieto da Silva, Research Associate,
MARE – Centro de Ciências do Mar e do Ambiente

October, 2018

Acknowledgments

I would like to express my gratitude to my institution supervisors, Professor Luís Soares and Professor Paulo Nunes for their great discussions, support, motivation and guidance throughout this process, which proved to be very important. Their constant positive outlook and good mood made the whole process a lot more enjoyable. I also wanted to note their availability, even in unreasonable time frames.

I'd also like to give a special thanks to supervisor Doctor Rui Prieto, who was responsible not only for the topic of the present work but was also a major influence throughout the development and writing of this work. His great help, patience, availability and enthusiasm were a big motivation throughout this process and absolutely invaluable.

Finally, I would like to deeply thank my family and friends that supported and pushed me throughout this phase. They were equally important and invaluable.

This thesis was supported by research project WATCH IT - Whale watching effects on sperm whales - disturbance assessment towards a sustainable ecotourism (ACORES-01-0145-FEDER-000057), co-funded by AÇORES 2020, through the FEDER fund from the European Union.

Resumo

Fotogrametria em câmera única é um procedimento bem estabelecido para recolher dados quantitativos de objectos através de fotografias. Em biologia, fotogrametria é frequentemente aplicada no contexto de estudos morfométricos, focando-se no estudo comparativo de formas e organismos. Nos estudos morfométricos são utilizados dois tipos de aplicação fotogramétrica: fotogrametria 2D, onde são utilizadas medidas de distância e ângulo para quantitativamente descrever atributos de um objecto, e fotogrametria 3D, onde são utilizadas coordenadas de referência de forma a reconstruir a verdadeira forma de um objeto. Apesar da existência de uma elevada variedade de *software* no contexto de fotogrametria 3D, a variedade de software concebida especificamente para a a aplicação de fotogrametria 2D é ainda muito reduzida. Consequentemente, é comum observar estudos onde fotogrametria 2D é utilizada através da aquisição manual de medidas a partir de imagens, que posteriormente necessitam de ser escaladas para um sistema apropriado de medida. Este processo de várias etapas é frequentemente moroso e requer a aplicação de diferentes programas de *software*. Além de ser moroso, é também susceptível a erros, dada a natureza manual na aquisição de dados. O presente trabalho visou abordar os problemas descritos através da implementação de um novo *software* multiplataforma capaz de integrar e agilizar o processo de fotogrametria presentes em estudos que requerem fotogrametria 2D. Resultados preliminares demonstram um decréscimo de 45% em tempo de processamento na utilização do software desenvolvido no âmbito deste trabalho quando comparado a uma metodologia concorrente. Limitações existentes e trabalho futuro são discutidos.

Palavras-Chave: Fotogrametria, Morfometria, Software, Biologia, Medição, Fotografia, Aplicação.

Abstract

Single-camera photogrammetry is a well-established procedure to retrieve quantitative information from objects using photography. In biological sciences, photogrammetry is often applied to aid in morphometry studies, focusing on the comparative study of shapes and organisms. Two types of photogrammetry are used in morphometric studies: 2D photogrammetry, where distance and angle measurements are used to quantitatively describe attributes of an object, and 3D photogrammetry, where data on landmark coordinates are used to reconstruct an object true shape. Although there are excellent software tools for 3D photogrammetry available, software specifically designed to aid in the somewhat simpler 2D photogrammetry are lacking. Therefore, most studies applying 2D photogrammetry, still rely on manual acquisition of measurements from pictures, that must then be scaled to an appropriate measuring system. This is often a laborious multi-step process, on most cases utilizing diverse software to complete different tasks. In addition to being time-consuming, it is also error-prone since measurement recording is often made manually. The present work aimed at tackling those issues by implementing a new cross-platform software able to integrate and streamline the photogrammetry workflow usually applied in 2D photogrammetry studies. Results from a preliminary study show a decrease of 45% in processing time when using the software developed in the scope of this work in comparison with a competing methodology. Existing limitations and future work towards improved versions of the software are discussed.

Keywords: Photogrammetry, Morphometry, Software, Biology, Measurement, Photography, Application.

Table of contents

List of tables

List of figures

List of abbreviations and acronyms

APP – Application

BMP – Bitmap (image file)

Exif – Exchangeable Image File Format

GC – Garbage Collector

GIF – Graphics Interchange Format

GIMP - GNU Image Manipulation Program

IFD – Image File Directories

I/O – Input/Output

ISO – International Organization for Standardization

JEIDA – Japan Electronic Industries Development Association

JFIF – JPEG File Interchange Format

JPEG – Joint Photographic Experts Group

MARE – Marine and Environmental Sciences Centre

MVC – Model-view-controller

MVP – Model-view-presenter

MVVM – Model-view-viewmodel

PC – Personal Computer

PNG – Portable Network Graphics

RDF – Resource Description Framework

TIFF – Tagged Image File Format

XMP – Extensible Metadata Platform

XML – Extensible Markup Language

Chapter 1 – Introduction

1.1. Context

The quantitative analysis of size and shape is essential for several branches of science, and comprises a set of measuring and analytical methods, collectively grouped in a field referred as Morphometrics (Elewa, 2010; Reyment, 1996, 2010; Rohlf, 1990). The origin of the term Morphometrics is attributed to Blackith (1957), however the field started being developed much earlier, with pioneering works being published by the end of the 19th century (e.g., Weldon, 1890). Morphometrics is deeply intertwined with biological studies on phylogeny, ecology, and physiology, but it also progressively found applications in geomorphology, medicine, anthropology, behavioral sciences and forensics (Elewa, 2010). Morphometric methods allow the description and comparison of shapes of organisms or objects in a standardized manner, allowing for investigations on variability due to distribution, development stage, gender, genetic and environmental effects, among others.

Traditional morphometrics has made use of (and in fact was partially on the basis for the development of) multivariate statistical methods, such as principal component analysis (PCA), cluster analysis, and similar methods, over data in the form of matrices of distance and angle measurements (Jensen, 2003; Rohlf & Marcus, 1993). This approach is still very useful for a plethora of applications; however, it has the drawback of not allowing the reconstruction of the original form from the measurements that are usually taken. Beginning in the mid-1980's, in part because of growing scientific needs, but also due to the unprecedented progress of computational power, a 'revolution' in Morphometrics took place, through new methods that enabled defining shape as a set of distances among landmark coordinates and the geometric information about their relative positions (Jensen, 2003; Rohlf & Marcus, 1993). These new methods utilizing landmark and outline analyses have been termed Geometric Morphometrics to differentiate from the traditional Morphometrics methods that were already well established (Adams, Rohlf, & Slice, 2004). The present work focuses in techniques related to the former case and, as such, hereon all mentions to Morphometrics and morphometry methods relate to traditional Morphometrics, unless otherwise stated.

1.2. Motivation and relevance

Despite the dramatic evolution seen in the field of Morphometrics, the process of recording morphometric data is often resource- and labor-intensive and can also be time-consuming when no appropriate tools are available (Adams, Rohlf, & Slice, 2013). Morphometry studies often involve the collection of multiple measurements from a representative sample of individuals in a population, which normally translates in at least few, to tens or even hundred thousands measurements taken in the course of a single study (Kocovsky, Adams, & Bronte, 2009; Rising & Somers, 1989).

Semi-automatic measuring equipment, linked to a computer, handheld PC, or with inboard processing capability, started being introduced in the early 1980's to side-step the laborious and error-prone process of transcribing measurement hand notes to a digital database (e.g., Morizur, Ogor, & Lespagnol, 1994; Sprules, Holtby, & Griggs, 1981). However, these equipments rely in the manipulation of the object or organism being measured, which is not always practical, or overall feasible. For example, capturing and measuring many marine organisms is often impossible due to logistical constraints related to the environment in which they live or to the sheer size of the animals (large whales being a paradigmatic example). Similarly, very small (microscopic) organisms and objects cannot be easily manipulated in order to obtain measurements. For those cases, other methods of measurement must be employed, one of the most prominent being photogrammetry.

Photogrammetry is a technique that makes use of photography to obtain measurements of physical objects, by scaling measurements taken directly on the photographic reproduction of an object, using basic geometric principles. Using the information stored in a photography to obtain measurements of objects was first proposed by François Arago, when discussing the applications of the (then) new daguerreotype technology (an early photographic process), during an address to the French Academy of Sciences (Arago, 1839). Since then, photogrammetry has become a well-established measuring procedure, being widely used in geodesy, but also in other fields ranging from cell biology to astrophysics (Kraus, 2011; Luhmann, Robson, Kyle, & Harley, 2006).

Photogrammetry techniques are diverse but the most common fall in two broad categories, aerial and terrestrial, which in turn can be either, close- (when camera to object distance is less than around 300 meters) or long-range (when camera to object distance is more than around 300 meters) (Luhmann et al., 2006). Micro photogrammetry (sometimes also termed 'macro photogrammetry' due to the type of lenses used) is a

special case of close-range photogrammetry, when the image scale >1 (Luhmann et al., 2006).

In the context of biological research, when photogrammetric techniques are used they tend to belong overwhelmingly to close-range photogrammetry and the basic procedure can be broken down in to three generalized steps:

1. Gathering proper and relevant photographs of the body structures meant to be measured;
2. Measuring the relevant body structures on the photographs, using either analogic or digital tools with appropriate resolution;
3. Scaling from image units to a relevant measuring system, such as metric or imperial, from known scaling equations.

As with direct morphometry, where measurements are taken directly from the specimens, in photogrammetry-aided studies making and recording measurements is usually also labor-intensive when done manually. Unfortunately, despite advances in automation of measurement acquisition in close-range photogrammetry, there is a number of situations where such automation is not possible, and the use of manual measurement is needed.

As mentioned earlier for direct morphometry, not only taking thousands of manual measurements from pictures can become one of the most time-consuming parts of any morphometry study, it also usually implies transcribing those measurements to a digital database, which adds to the lengthy process of data acquisition and preparation prior to analysis. Additionally, data errors from transcription to database can compromise data quality (W. Kim, Choi, Hong, Kim, & Lee, 2003). In fact, data is usually corrupted during data entry by humans, either due to typographical errors or erroneous interpretation of written notes, and is still one of the most common sources of data-quality issues in databases (Hellerstein, 2008).

It becomes clear that photogrammetry-aided morphometry studies can benefit from software that aid in capturing and transcribing data in an automatic or semi-automatic way. However, it is often the case that existing software solutions are either too expensive or are not appropriate for having been designed for a totally different application and/or not contemplating all steps in a workflow. Well designed software should aid during several stages of the workflow, decreasing both processing time and human error by

reducing complexity and automating data transcription (Gentleman et al., 2004; Hellerstein, 2008).

The present work aimed at creating a software package to tackle those goals while ensuring enough flexibility to be useful in a wide-range of morphometric studies. Researchers from the Azores Whale Lab, associated with the Marine and Environmental Sciences Centre (MARE), helped with the development of the software by iteratively testing it and by providing feedback about the user experience.

Morphometric data from whales is essential for several lines of research developed at the Azores Whale Lab. Nevertheless, due to their size, capturing whales is unfeasible and, consequently, photogrammetry is a preferred way to obtain relevant morphological data from different individuals. Photogrammetry provides additional benefits, such as larger sample sizes due to cost-effectiveness, increasement of safety to the researchers, and less disturbance to the whales (J. W. Durban, Fearnbach, Barrett-Lennard, Perryman, & Leroi, 2015).

1.3.Investigation objectives

The general objective is to explore, formulate and implement a solution to simplify and expedite the retrieval of morphometric data of animals from photographs with results comparable to current solutions. The proposed and implemented solution, therefore, must be an easier and faster way to retrieve morphometric data than current procedures. It will be compared with the current photogrammetry procedure employed by the researchers at Azores Whale Lab.

It is important to note that although direct comparisons will be made against the Azores Whale Lab photogrammetry procedure, the solution should still be general enough to be used in different photogrammetry contexts and applications. It would also be ideal if the solution is cross-platform, or at least capable of easily being ported into other platforms.

1.4.Methodology

As mentioned above, the solution proposed in the scope of this work was developed with the cooperation of researchers at Azores Whale Lab and direct contact with them was crucial, namely in gathering feedback about design choices, usability and bugs. For development, a Kanban based agile methodology was used (Sugimori, Kusunoki, Cho, &

Uchikawa, 1977), where continuous releases and adjustments were made based on feedback. Ideally, this would span a one- or two-week cycle of development to testing, but in reality, it was not as regular and it varied from one week to one month.

The development releases of the program were published on the GitHub platform. There were several reasons for this, namely:

- It allows public discussion and feedback through the use of the issues feature;
- It provides a way to label each issue which provides additional organization. Labels such as ‘bug’, ‘duplicate’, ‘enhancement’ and priority related labels were created and used to help organize and prioritize the development work;
- It provides an easy and intuitive way to track the progress of each issue, through the use of a Kanban board, where each issue can be seen akin to a ticket;
- It provides a wiki platform, where it is possible to explain and detail how the program works and how it can best be used;
- It provides a way to publish and track different releases of the program.

Final analysis and comparison in usability and usage time between current solutions and the new solution was also done with the cooperation of researchers at the Azores Whale Lab. The comparison details, including the data collection process, results and limitations is further discussed in Chapter 4.

1.5. Dissertation structure and organization

This document is organized in five different chapters describing the different phases until its conclusion.

The first chapter introduces the dissertation theme, the motivation, its objectives and the document structure.

The second chapter expands the relevance, current solutions and theoretical framework of the photogrammetry procedure and each of its steps.

The third chapter describes in detail the proposal and implementation of an integrated system capable of solving the presented challenges that were identified in previous chapters.

The fourth chapter presents a comparison between previous solutions and the proposed system through the analysis of quantitative and qualitative results.

The fifth and last chapter presents the study conclusions, recommendations, limitations, possible contributions to the scientific community and future research.

Chapter 2 – Photogrammetry theoretical framework

2.1. Introduction

This chapter is composed of five Sections and describes the theoretical framework behind the photogrammetry process and its steps. The first Section introduces and describes the chapter structure.

The second Section explores the different applications of photogrammetry, as well as its relevance to biological sciences. It further delineates the general process of photogrammetry and describes a specific example within the field that will be used as reference throughout this work. Finally, relevant software and existing solutions are presented.

The third, fourth and fifth Sections describe the relevance and theoretical framework of relevant steps in the photogrammetry process noted in the second Section. The third Section relates to the specific step of camera calibration and image distortion correction. The fourth Section relates to image metadata extraction. Finally, the fifth Section relates to measurement acquisition and unit conversion, also referred as scaling.

2.2. Computer-aided photogrammetry

2.2.1. Relevance and current solutions

As has already been pointed in Chapter 1, photogrammetry has a wide range of applications, with examples in many fields, including microscopy (e.g., Boyde & Ross, 1975), archaeology and paleontology (e.g., Fussell, 1982; Mallison & Wings, 2014), anthropology and forensics (e.g., Grip, Grip, & Morrison, 2000; Milliet, Delémont, & Margot, 2014), astronomy (e.g., Googe, Eichhorn, & Luckac, 1970), industry (e.g., Fraser & Brown, 1986), geodesy (e.g., Kraus, 2011), among many others.

The use of photogrammetry is also widespread in biological and ecological studies and can be motivated by conservation and ethical concerns (e.g., Whitehead & Gordon, 1986; Whitehead & Payne, 1981), to speed-up obtaining and processing data on studies involving a large number of individuals or over large areas such as in forestry (e.g., Ivosevic, Han, & Kwon, 2017; Zou et al., 2014), or to obtain data on organisms that are difficult to locate, manipulate, or inaccessible in other ways (e.g., Breuer, Robbins, & Boesch, 2007; Deakos, 2010; John W. Durban et al., 2016; Letessier, Juhel, Vigliola, & Meeuwig, 2015). Another incentive for using photogrammetry is its relative low cost

compared with direct measurement in many situations, which is often made clear in the titles of resulting publications, by the use of terms such as ‘Inexpensive’ and ‘Low-cost’ (e.g., Stephen M Dawson, Bowman, Leunissen, & Sirguey, 2017; S. M. Dawson, Chessum, Hunt, & Slooten, 1995; Letessier et al., 2015; McFall, Shepard, Donaldson, & Hulbert, 1992).

Apart from small variations in the way pictures are obtained, the workflow for photogrammetry studies in natural sciences (e.g., Stephen M Dawson et al., 2017; Deakos, 2010; Galbany et al., 2016; Jadejaroen, Hamada, Kawamoto, & Malaivijitnond, 2015; Meise, Mueller, Zein, & Trillmich, 2014; Rothman et al., 2008; Shrader, Ferreira, & Van Aarde, 2006) is very similar, normally involving several steps, namely:

1. Correction of distortion within the taken photographs, using camera calibration;
2. Obtaining data to enable scaling pictures either by:
    a. Placing a scale in the same plane as the specimens (which can be achieved by the projection of parallel laser beams, placing a scale bar or using another object of known size), or
    b. Measuring the distance from the camera focal plane to the specimen plane.
3. Measuring the relevant structures in the photographs, usually by utilizing image editing software;
4. Exporting measurements to a digital database;
5. Making calculations and statistical analyses using spreadsheets and statistical analysis software.

The exact number of steps may vary depending on the nature of the study and the methodologies utilized.

2.2.2. Azores Whale Lab photogrammetry process

An example of a photogrammetry workflow is the protocol used by researchers at Azores Whale Lab at the beginning of this project, which involved the following specific steps (R. Prieto, personal communication, 2017):

1. Photograph acquisition of relevant body structure(s) of whale individual(s) through two methods (see Figure 1):

   a. Aerial photoshoots, which makes use of unmanned aerial vehicles (hereby termed ‘drones’) to photograph the full whale’s body while the animal is at the surface;
   b. Boat photoshoots, which means that only part of the whale can be photographed. In the case of sperm whales, for example, the whale fluke is normally picked to be photographed as empirical allometric expressions that relate the fluke with the rest of the whale’s body exist (Jaquet, 2006).
2. View, selection and organization of acquired photographs using Irfanview (Skiljan, 2003);
3. Extraction of relevant metadata from the acquired image files using ExifTool (Harvey, 2013);
4. Correction of the camera’s lens distortion in the photographs for accurate measurements. This can be divided in two steps:
   a. Camera calibration, where different camera parameters are estimated to model the lens appropriately;
   b. Undistorting the photographs according to the model obtained in step 4a.
5. Measurement of the relevant anatomical structures in pixels using ImageJ (Abràmoff, Magalhães, & Ram, 2004; Schneider, Rasband, & Eliceiri, 2012); or GIMP (Anonymous, 2015);
6. Conversion of measurements from pixels to a conventional measuring system using a custom Microsoft Excel spreadsheet;
7. Solving allometric expressions, if applicable, using the same custom Excel spreadsheet.

Relevant data extracted from this process are saved to a spreadsheet, which is also why, for convenience, the same spreadsheet is used in step 6 and 7.

Figure 1 – An aerial photograph of a full whale's body (left) and a photograph taken from a boat photoshoot (right) (Azores Whale Lab, n.d.).

One aspect that is immediately apparent from the description of the process above is that there is a strong reliance on different types of software to perform diverse but interlinked tasks. Consequently, information must be passed through each software tool manually. As mentioned, this increases processing and data analysis time, as well as increasing the chances of human error.

2.2.3. Photogrammetry software review

Although some software tools specifically designed for photogrammetry do exist, the available solutions are usually proprietary, with expensive licensing fees. Additionally, the development of photogrammetric software, independently of type of license (commercial or free), has focused mainly on multi-image 3D reconstruction of objects. Currently, there are not many publicly available solutions focused on single-camera photogrammetry, although there are some software packages that can perform some of the tasks listed above.

Quite recently, a study on the accuracy of whale morphometry data obtained using drone images, resulted in the creation of a toolset constituted by two programs written in MATLAB and a script in the statistical programming language R to aid in single-camera morphometry of whales, that can be useful also to photogrammetry studies on other similar organisms (Burnett et al., 2018). Notwithstanding, although aimed specifically for single-camera photogrammetry studies, the programs created by Burnett et al. (2018) are specifically designed for aerial photogrammetry using drones and constrain the measurements to a set of pre-defined metrics that were specifically designed for whale morphometry, severely restricting its applicability.

Consequently, there is clearly a need for a software tool that enables performing part or all the steps detailed above and is flexible enough to be useful for studies with different methodologies and objectives. The present work aims to tackle those objectives by designing a cross-platform software, implemented in Java, specifically aimed at single-camera, close-range photogrammetry, with a simple interface and a set of tools that cover most needs in this type of studies.

In the following Sections, the main different challenges and requirements in the photogrammetry workflow are described and explored in detail.

2.3. Camera calibration

2.3.1. Relevance and current solutions

In theory, if cameras were to be manufactured perfectly (i.e., fully described by the classic pinhole model), then all it would be needed to start measuring objects from photographs would be the focal length, pixel size and the distance from the object. However, real world cameras are not perfect. Some may contain non-square pixels, some amount of skew and all will contain varying degrees of lens distortion. As such, it is important to account for these variables to retrieve accurate and meaningful measurements. This is also the reason why camera calibration receives special attention throughout this work.

Some types of lenses have very distinct and noticeable radial distortions (Jedlička & Potůčková, 2007) as can be seen in Figure 2. Fisheye lenses, for example, are normally associated with barrel distortion, referred as negative displacement, which has the effect of the image appearing to be mapped around a sphere. This is because the field of view of these lenses is wider than the sensor size. There is also pincushion distortion, referred as positive displacement, which is often associated with telephoto lenses and narrower lenses. Finally, a mix of both can also occur and is termed mustache distortion.

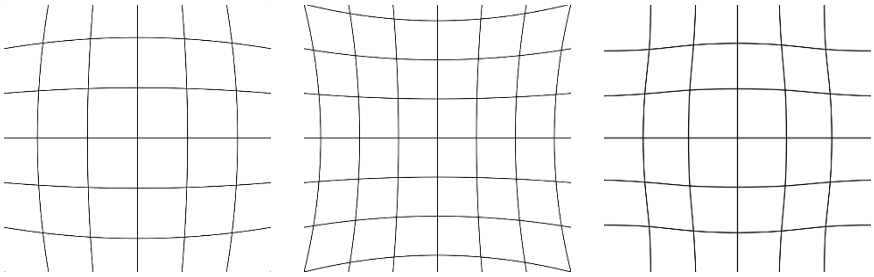


Figure 2 - Types of typical radial distortions. Barrel distortion (left), pincushion distortion (middle) and mustache distortion (right) (WolfWings, 2010).

There are several solutions available for calibration purposes, namely Camera Calibration Toolbox for Matlab (Bouguet, 2000), DLR CalLab (Strobl, Sepp, Fuchs, Paredes, & Arbter, 2010) and Calib3V (Balletti, Guerra, Tsioukas, & Vernier, 2014). Calib3V is the solution used in the Azores Whale Lab procedure with satisfactory results. It is able to obtain camera calibration parameters and produce new undistorted images from those same parameters.

For developers, there are two prominent libraries that allow implementing calibration into their own software, namely OpenCV (Bradski & Kaehler, 2000) and BoofCV (Abeles, 2012). OpenCV is a very well-known and mature open source computer vision library implemented in C++, while BoofCV is a more recent and not as mature contender implemented in Java.

OpenCV was picked to be used in the current project for its maturity and almost complete calibration functionalities, able to calibrate both low and high distortion lenses. Despite being implemented in C++, it also provides a Java wrapper that normally provides sufficient compatibility in most use cases.

2.3.2. Camera calibration principles

Camera calibration consists in estimating the camera's intrinsic and extrinsic parameters. Intrinsic parameters characterize the camera and lens, remaining unchanged if camera/lens setup is unchanged, while the extrinsic parameters are transformations that map world coordinates to camera coordinates, changing for each image. Camera calibration has several applications, such as estimating 3D structures based on camera motion, measuring, estimating depth and position of objects.

There are different techniques to calibrate a camera (Qi, Li, & Zhenzhong, 2010). Traditional camera calibration techniques involve using known calibration points. Some

of those techniques use a pattern within a 3D object (e.g., a cube), others a pattern within a planar object. There are also auto-calibration techniques, where no known calibration points are needed, and instead use a sequence of images to track points of interest (Faugeras, Luong, & Maybank, 1992).

There are also different mathematical models used in calibration to describe mono and stereo lens cameras, as well as different models to describe normal lenses (i.e., typical lenses that reproduces a field of view natural to the human eye), and wide-angle lenses (e.g., fisheye lenses often present in action cameras).

Every solution presented in Section 2.2.1 has as their main focus planar based camera calibration, although some provide support for other calibration techniques. Planar based camera calibration was introduced in Zhang (2000) and consists on estimating the camera's parameters by analyzing photographs of a planar calibration pattern at different random angles, as can be seen in Figure 3. Because of its simplicity to the user, it is also the technique that was chosen to be implemented in the proposed solution.

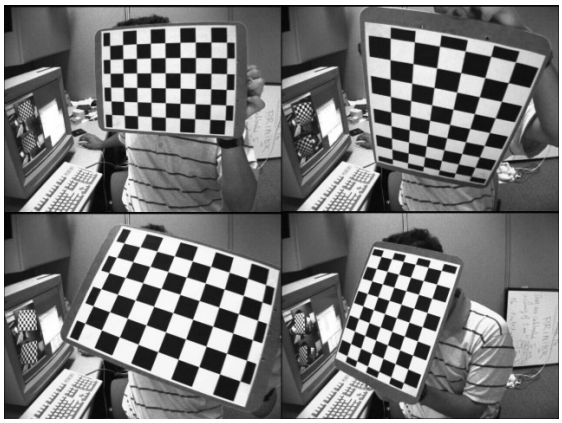


Figure 3 – A set of photographs of a planar calibration pattern at different angles to be used for camera calibration[1].

Planar based camera calibration techniques can be broken down into three steps:

1. **Calibration image acquisition**: Printing and placing a pattern on a rigid planar surface, and consequently taking different photographs of the pattern at different angles;
2. **Point detection/extraction**: Extracting point coordinates from the calibration images, be it corners from the chessboard pattern, centers from the circle grid pattern, or other. This step is further discussed in Sections 2.3.3, 2.3.4 and 2.3.5;

[1] Retrieved from https://github.com/opencv/opencv/tree/master/samples/data *(left01.jpg to left14.jpg)*

3. **Camera parameter estimation**: Based on extracted point coordinates, estimate the camera intrinsic, extrinsic and other related parameters, such as distortion coefficients. This step is further discussed in Sections 2.3.6 and 2.3.7;

2.3.3. Point extraction – Chessboard pattern

One very popular way to derive known calibration points to be used for calibration is to use a chessboard pattern. OpenCV implements the chessboard pattern finding algorithm in a function called findChessboardCorners(), and is based on the work of Vladimir Vezhnevets, Philip Gruebele, Oliver Schreer and Stefano Masneri[2]. An example of a detected chessboard pattern can be seen in Figure 4. It uses a graph of connected quadrilaterals (quads) to detect calibration points, i.e., corners on the chessboard, and, according to the source code[1], implements the following simplified steps:

1. Converting the original image to a binary image, where the threshold is decided based on an analysis of the image histogram (pixel intensity);
2. If the CALIB_CB_FAST_CHECK flag is set, it attempts to solve the degenerate case where no pattern exists in the image, saving computation time. It does this by:
    a. Applying the opening morphological operation (erosion followed by dilation), to help split the chessboard squares so they can be properly recognized as quadrilaterals;
    b. Retrieving every quadrilateral in the image using a combination of a threshold function and findContours function, which uses the algorithm described in (Suzuki, 1985);
    c. Using a flood fill style algorithm, it checks if there are many quadrilaterals with similar sizes. If there are, continues to step 3. If there are not, the function finishes.
3. Dilation morphological operation on the binary image calculated on step 1 to separate quadrilaterals;
4. Applying a border around the image to detect possible clipped quadrilaterals;
5. Using findContours() function described in step 2.band applying the Ramer-Douglas-Peucker algorithm to simplify identified contours and find quadrilaterals (Douglas & Peucker, 1973; Ramer, 1972);

[2] Source code available in https://github.com/opencv/opencv/blob/master/modules/calib3d/src/calibinit.cpp

6. Filter any identified contours that do not represent connected quadrilaterals with acceptable size until the proposed number of input corners are found;
7. It calls cornerSubPix() function to further refine the corner locations with sub-pixel accuracy, using a gradient-based optimization.

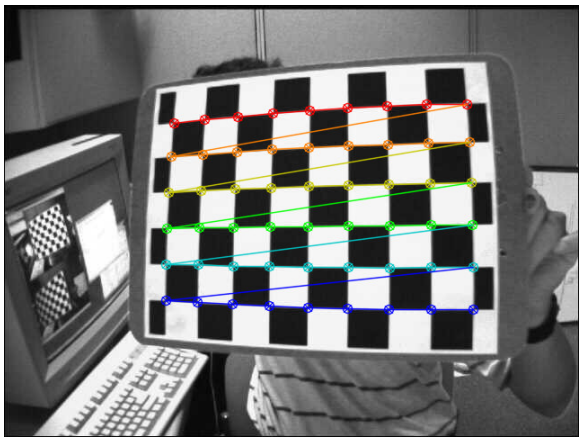


Figure 4 – Representation of detected corners on a chessboard pattern[3].

The function works sufficiently well but it has a few constraints. The authors note the following constraints: the chessboard has to have odd x even (or vice-versa) number of squares (e.g., 9x6), and the board itself should have at least a square sized white border. Additionally, the user must input the number of squares (rows and columns) of the chessboard for the algorithm to work.

The function also has some issues, namely related to the corner detection rate and corner position accuracy. For the corner detection rate, if the projection angle is too large or if the area of the square is too small, then sometimes the function fails. For the corner position accuracy, as noted in Datta, Kim, and Kanade (2009), the refinement step assumes that square gradients will be orthogonal to the edges. This is seldom the case, since usually the calibration pattern will be distorted, and not in the fronto-parallel pose. This causes accuracy issues with the detected corners which directly affects calibration results.

This function employed by OpenCV relies on the geometry of the chessboard itself. However, throughout literature, there are several other suggestions to deal with detecting chessboard corners. From the traditional Harris corner detector (Harris & Stephens, 1988), smallest univalue segment assimilating nucleus (SUSAN) (Smith & Brady, 1997)

[3] Retrieved from https://docs.opencv.org/ref/master/d9/dab/tutorial_homography.html

to Hessian matrix-based methods (Chen & Zhang, 2005; Liu, Liu, Cao, & Wang, 2016) with comparable results. There are also solutions that eliminate the need for the user to input the chessboard rows and columns number (De la Escalera & Armingol, 2010). There are even proposals to directly improve the OpenCV algorithm in the cases of images with low resolution, high distortion (as is the case of wide-angle lenses) and blurriness (Rufli, Scaramuzza, & Siegwart, 2008).

2.3.4. Point extraction – Circle grid pattern

The OpenCV function for the circle pattern finding algorithm called findCirclesGrid() can, depending on the parameters, detect either symmetric or asymmetric circle grid patterns. Like the chessboard corner detector, the user needs to input the number of rows and columns. The algorithm makes use of a simple blob detector to extract the center of dark circular blobs (using the findContours function) and creating a neighboring graph to find the pattern, filtering outliers. An example of a detected circle grid pattern can be seen in Figure 5.

The use of a circle grid pattern has been argued that it may yield better precision than the chessboard pattern (Xiao & Fisher, 2010). Furthermore, it was also noted in Balletti et al. (2014) that the circle grid pattern was more resistant to defocus effects than the chessboard.

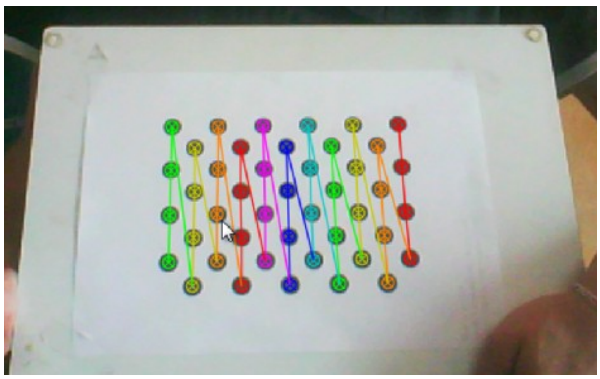


Figure 5 – Representation of found circle centers on a circle grid pattern[4].

[4] Retrieved from https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

2.3.5. Point extraction – Other calibration points methods

There are other patterns/techniques that deserve attention. The use of concentric circles has been shown to potentially yield even better point accuracy and consequent calibration results (Datta et al., 2009; J.-S. Kim, Gurdjos, & Kweon, 2005; J.-S. Kim, Kim, & Kweon, 2002; Vo, Wang, Luu, & Ma, 2011). Very recently, the use of deltille grids (see Figure 6) with monkey saddle fitting has also been shown to surpass significantly both the detection rate and corner accuracy of the chessboard pattern, even in highly distorted images (Ha, Perdoch, Alismail, Kweon, & Sheikh, 2017). Machine learning may also have an impact in this field: the use of convolutional neural network (CNN) is shown to potentially be able to generalize point extraction under several degrees of image degradation (Donné, De Vylder, Goossens, & Philips, 2016). OpenCV calibration solution was still the chosen to be implemented as it has been proven, in terms of accuracy, sufficient in most use cases (Balletti et al., 2014), and because integration is significantly simpler, since the code is freely available and compatible with Java projects.

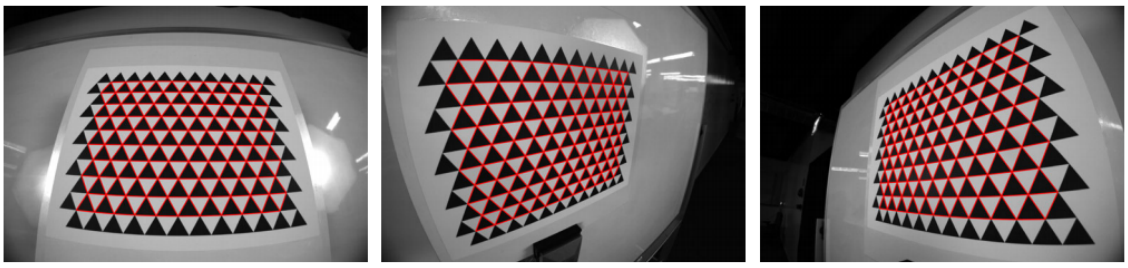


Figure 6 – Representation of found corners on a deltille grid pattern (Ha et al., 2017).

2.3.6. Calibration – Pinhole model

To calibrate a camera after extracting the pattern points, OpenCV provides a function called calibrateCamera(). Most of OpenCV calibration routines are directly ported from Camera Calibration Toolbox for MATLAB (Bouguet, 2000) with some added functionalities, which are heavily inspired by the procedures and models of Zhang (2000) and Heikkila and Silven (1997). Those are explained below, with notation matching a combination of Zhang's paper, Bouguet implementation and Hartley and Zisserman's book (Hartley & Zisserman, 2003).

For the pinhole model, let a 2D point be defined as $p = [u, v]^T$ and a 3D point defined as $P = [X, Y, Z]^T$. Further, let $\tilde{p}$ and $\tilde{P}$ be augmented vectors of p and P respectively, by

adding 1 as the last element (the so called homogeneous coordinates). The relation of a 3D point P and its image projection p is given as:

$$c\tilde{p} = K[R\ \ t]\tilde{P}, \qquad \text{with } K = \begin{bmatrix} F_x & s & u_0 \\ 0 & F_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

where c is an arbitrary scale factor; R and t are defined as a 3×3 rotation matrix and a 3×1 translation column-vector respectively, and relate the world coordinates to the camera coordinates; F_x and F_y denote focal length in both x and y axes, but can abstractly be referred as scale factors; (u_0, v_0) are the coordinates of the principal point, which is the point from where the focal length is measured, relative to the image plane origin and s is the skew between both images axes. (R, t) are also referred as the extrinsic parameters and K is referred as the camera intrinsic matrix. P, in this case, will signify the calibration points detected previously in 3D space.

In a true pinhole model, F_x and F_y are equal, which would result in the expected square pixels. However, in practice they may differ for different reasons, such as calibration errors, manufacturing flaws (e.g., in the camera sensor), non-uniform post-process scaling, and others (Simek, 2013). The principal coordinates (u_0, v_0) are normally the center coordinates of the image but may also fluctuate for similar reasons presented before. However, because of modern-day cameras manufacturing quality, sometimes the focal length, the skew and/or the principal point can be constrained when calibrating.

The model plane can also be assumed to lie on $Z = 0$ of the world coordinate system without loss of generality. Each column of the rotation matrix R can also be notated by r_i where i is ith column. Equation (1) can now be rearranged as follows:

$$c\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\ [r_1\ \ r_2\ \ r_3\ \ t]\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K\ [r_1\ \ r_2\ \ t]\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Note now that $\tilde{p} = [u, v, 1]^T$ and $\tilde{P} = [X, Y, 1]^T$ are now related by an homography H:

$$c\tilde{p} = H\tilde{P} \text{ with } H = K\ [r_1\ \ r_2\ \ t]$$

Further, H can be denoted as $H = [h_1\ \ h_2\ \ h_3]$. With this we now have:

$$[h_1\ \ h_2\ \ h_3] = \lambda K[r_1\ \ r_2\ \ t],$$

where λ denotes an arbitrary scalar. We can now constrain the intrinsic parameters, since we know that r_1 and r_2 are orthonormal. Consequently, it is known that the dot product of these two vectors will be zero and that both are of the same length. As such, we have:

$$h_1^T K^{-T} K^{-1} h_2 = 0$$
$$h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2$$

The next two steps, described in Zhang (2000), involve calculating the homography and consequently estimate both the intrinsic and extrinsic parameters using every input image. OpenCV/Bouguet's Toolbox implementation derive from Zhang's method here, but use vanishing points to estimate the focal length. Their initial estimation also ignores any lens distortion coefficients but it is considered in the next steps.

Before continuing, it is relevant to note that OpenCV/Bouguet internally uses a different intrinsic model than the one presented in (1). To the user, however, the intrinsic matrix is still represented for both input and output of their functions as (1). Internally, OpenCV model follows the one used in Heikkila and Silven (1997) very closely and is represented as follows.

By abuse of notation, consider that $P = [X, Y, Z]^T$, but now that P already represents a point in the camera coordinates, after being transformed from world coordinates through the extrinsic parameters (rotation and translation). The corresponding image coordinates (u, v) are now given as:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f_x X \\ f_y Y \end{bmatrix}$$
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s\tilde{u} \\ \tilde{v} \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}, \quad (2)$$

Real world cameras are not true pinhole models, as they always exhibit distortion (even if very minor). Radial distortion is the most predominant and, in fact, Zhang's model only considers it. Heikkilä's, however, also considers tangential distortion. There are other types of distortions considered throughout literature, such as linear distortion (Melen, 1996) and prism distortion (Weng, Cohen, & Herniou, 1992), but these are not considered in OpenCV/Bouguet's implementation as they are not as relevant. The models can now be reformulated to account for both radial and tangential distortions.

Radial distortion is normally represented in the following form:

$$\begin{bmatrix} \delta u^{(r)} \\ \delta v^{(r)} \end{bmatrix} = \begin{bmatrix} \tilde{u}(k_1 r_i^2 + k_2 r_i^4 + \dots) \\ \tilde{v}(k_1 r_i^2 + k_2 r_i^4 + \dots) \end{bmatrix},$$

where $k_1, k_2, \dots, k_i$ are coefficients for the radial distortion, and $r_i = \sqrt{\tilde{u}^2 + \tilde{v}^2}$. In the OpenCV implementation, generally only two or three radial distortion coefficients are

considered, but it has the possibility to go up to six using a rational model for wide-angle lenses (refer to Section 2.3.7).

In the case of tangential distortion, it is normally represented as follows:

$$\begin{bmatrix} \delta u^{(t)} \\ \delta v^{(t)} \end{bmatrix} = \begin{bmatrix} 2p_1\tilde{u}\tilde{v} + p_2(r_i^2 + 2\tilde{u}^2) \\ 2p_2\tilde{u}\tilde{v} + p_1(r_i^2 + 2\tilde{v}^2) \end{bmatrix},$$

where p_1 and p_2 are coefficients for the tangential distortion. Both distortions can now be combined with the model in (2) to describe a projected point in the image plane as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} s\left(\tilde{u} + \delta u^{(r)} + \delta u^{(t)}\right) \\ v + \delta v^{(r)} + \delta v^{(t)} \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{3}$$

The last steps of calibration involve refining the estimated parameters through the use of maximum likelihood estimation. This step minimizes the reprojection error using the Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963). Bouguet only includes the distortion coefficients in this refining stage.

The reprojection error is the Euclidean distance between a feature point (i.e., a point extracted in the original image) and the projected point. For a single image with n calibration points, it is calculated as follows.

$$\sqrt{\frac{\sum_{i=0}^{n} d\left(\tilde{p}_i,\ \tilde{P}_i\right)^2}{n}} \tag{4}$$

The reprojection error, therefore, can also be interpreted as to how well the calibration model fits the given calibration images.

2.3.7. Calibration – Fisheye model

The mathematical camera model that OpenCV uses in all their main calibration functions does not generalize well enough for all types of camera lenses. If image correction with said model is attempted on photographs obtained with camera lenses that have a high level of distortion, e.g., wide angle fisheye lenses, results similar to what can be observed in Figure 7 are obtained. While the pixels close to the principal point can be considered corrected, a high level of distortion on the outer edges is observed.

Figure 7 – Original image (left) and the "undistorted" image using the regular Pinhole model (right).

OpenCV, however, does provide two ways to handle this degree of distortion. The first is by using a rational model detailed in Claus and Fitzgibbon (2005). This model explores the mapping of image coordinates to appear in quadratic polynomials, mapping $\tilde{P}$ to a six-dimensional space in a 6-vector of monomials.

The other way that OpenCV provides to handle this degree of distortion is by using the model described in Kannala and Brandt (2006). This approach tries to accurately model the geometry of real camera lenses and abandons the pinhole model, generalizing a camera model that is argued to be suitable for both omnidirectional and conventional cameras. Correcting an image using this latter model results in a more accurate result, as can be seen in Figure 8.

Figure 8 – Original image (left) and the corrected image using the Kannala model (right).

2.3.8. Lens distortion correction

After the calibration process is finished and the camera matrix is calculated, correcting an image is then trivial. Each pixel coordinates in the target undistorted image is mapped to the corresponding pixel coordinates in the original input image and the corresponding

value for each pixel is then interpolated, providing results similar to what can be observed in Figure 8.

In detail, consider that (u, v) now represents each pixel coordinates of the target undistorted image. The corresponding pixel coordinates in the original input image, (X, Y), can be found by inverting the mapping defined in equation (3), i.e., given (u, v), $P = (X, Y, 1)$ is calculated. This yields non-integer coordinates, so typically the four neighboring pixel values are bi-linearly interpolated. If there is no correspondence, the pixel is set to a black color.

2.4. Image metadata

2.4.1 Relevance and current solutions

Many types of files contain metadata information that provide information about the file itself, which is also true with image files. Many image file formats have metadata information that encode data such as image size, pixel information, etc. In modern day digital cameras, they also provide additional information about the images taken, such as camera model, ISO, aperture, exposure time, geolocation information and so on.

In this context, extracting metadata from images serves three different purposes:

1. **Record purposes**: It is useful to store basic information about the image, lens, camera, or location where the photograph was taken alongside with the measurements;
2. **Calculation purposes**: Some information is useful to perform the photogrammetry process. For example, in the case of a drone photoshoot, the distance between the camera and the object in the ground, i.e., altitude, is usually found in the metadata.

Additionally, it is also a good idea to maintain metadata integrity when modifying an image in any way. For example, in the case of lens distortion correction, the metadata information must be kept when saving the corrected image.

Exiftool (Harvey, 2013) is one of the most popular and mature metadata extraction software available. It exists both as a Perl library and a command-line tool. It supports a large amount of file types and formats, it is able to read, write and copy metadata information between files and also has a large list of features.

In the Java environment, the most stable and complete image metadata extraction library is the metadata-extractor developed by Drew Noakes (Noakes, 2002). It is able to extract all the different required formats of metadata, such as Exif, XMP and JFIF. It is also able to process a long list of image types, such as JPEG, PNG, BMP, GIF, and different camera raw formats. However, it is not able to write metadata to a file, which means it does not help with retaining metadata integrity.

Metadata integrity is fulfilled by the Apache Commons Imaging library ("Commons Imaging: a Pure-Java Image Library," 2017), which although does not support the same long list of formats and image types as metadata-extractor, is able to write metadata into a file. Unfortunately, it is not very mature, and it does not seem like there are any current developments.

2.4.2 Exif

For the recording and calculation purposes, we are mostly concerned with metadata information contained within the Exif and XMP standards. Details about the location, date, camera, its settings and distance indicators are especially relevant.

Exchangeable image file format, or Exif, is a popular standard that specifies image and audio files formats used by digital cameras and was developed by the Japan Electronic Industries Development Association (JEITA, 2002). It covers most of the relevant information, such as GPS information, date and time, camera settings, thumbnail, descriptions and copyright information.

Exif, in the context of image files, was originally designed to support the JPEG standard for compressed images, and TIFF for uncompressed images. Some raw formats can also include Exif metadata, such as CR2 (Canon Raw version 2). JPEG/Exif is a very common image format for digital cameras, while JPEG/JFIF (JPEG File Interchange Format) is common for storing and transmitting photographs over the World Wide Web.

In the case of a JPEG/Exif image, the structure is well defined. A JPEG file is split into a sequence of segments. One of those segments, called application marker (APP), contains application specific information and is where Exif metadata information resides (specifically in APP1). Exif follows a similar structure to the TIFF tag scheme and its size cannot exceed 64 Kbytes specified in the JPEG standard. It uses two file directories, called Image File Directories (IFD). The first IFD (0th IFD) contains information, i.e.,

attributes, about the image and the second (1st IFD) is normally used for the thumbnail image. A representation of this structure can be seen in Figure 9.

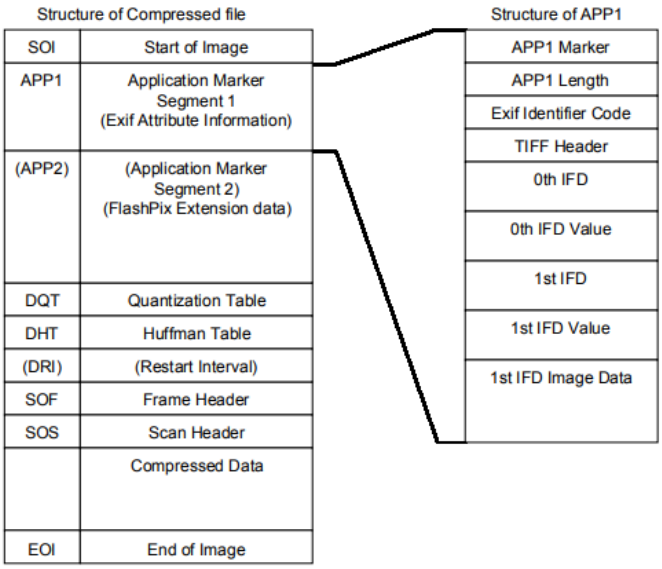


Figure 9 – Basic structure of a JPEG compressed file (JEITA, 2002).

Each metadata attribute is composed of four elements: tag, type, count and value offset. The element tag is a 2-byte identifier of the attribute; the type identifies the value type, e.g., short, long, etc.; the count is the number of values (and not of the bytes) and value offset is the offset from the TIFF header to the position where the value is stored.

2.4.3 XMP

Extensible Metadata Platform, or XMP, is an ISO standard developed by Adobe (2012) used for standardized and custom metadata in digital documents. In this context, it is especially relevant because it is able to store custom metadata information in images, that will not be stored in Exif. It is common to see external sensor information stored with XMP, e.g., distance, altitude, etc.

XMP normally uses Resource Description Framework (RDF) syntax (Adobe, 2012), which is written in Extensible Markup Language (XML), and the data model is also referred as a XMP packet. XML is a markup language that was designed to be both interpreted by machines and humans and examples of this language can be seen in Appendix B, C and D. RDF simply defines a vocabulary, so metadata can be encoded in

the XML language. Since it is based on XML, it does not allow binary data types. For JPEG files, it is also stored in the APP1 segment.

2.5. Measuring and unit conversion

Most general-purpose image editors are able to measure finite straight lines (i.e., distance between two points) in pixels, as is the case of GIMP and ImageJ, used by the Azores Whale Lab. Additionally, they are capable of measuring straight lines constrained by an angle. For example, when measuring the full body of a sperm whale, it is useful to first measure the length of the whale and then the width of the whale, where the width lines should be perpendicular to the length.

ImageJ (Abràmoff et al., 2004; Schneider et al., 2012) is, in fact, very often used in photogrammetry studies. It is a powerful, flexible and cross-platform software for image analysis, that admits user-written macros and plugins. It has gained wide acceptance in the scientific community, especially, but not restricted to, in microscopy and medical imaging (Schneider et al., 2012). In fact, ImageJ is often the program of choice to obtain and scale measurements in photogrammetry studies (e. g. Jadejaroen et al., 2015; Sadou, Beltran, & Reichmuth, 2014). Despite having extensive support and plugins aimed at a variety of research fields, its focus remains on image processing, and despite a large database of plugins (https://imagej.nih.gov/ij/plugins), currently there are no ImageJ plugin devoted to tackle the single-camera photogrammetry workflow outlined in Section 2.2.

Converting the measured lines lengths from pixels to a conventional unit, has two general techniques associated that are employed by the Azores Whale Lab:

1. **Reference based conversion**, where an object with known dimensions is present in the photograph and pixel size in the desired unit is consequently calculated. This is used in boat photoshoots, where a scale can be projected using parallel lasers over the animal's body;
2. **Distance based conversion**, where the ratio between focal length and pixel size of the sensor is related to the ratio between the distance to the object and the pixel size of the object. This is used in drone photoshoots, where the distance to the object is normally the altitude of the drone.

ImageJ accommodates unit conversion with the “Set scale” feature, where the user is able to set the ratio between the pixel distance and the desired unit of distance, however that ratio must be calculated by the user beforehand.

Chapter 3 – Integrated photogrammetry system proposal

3.1. Introduction

This chapter is composed of ten Sections and describes the proposal and implementation of the solution to the previously mentioned challenges in the photogrammetry process. The first Section introduces and describes the chapter structure.

The second Section defines the solution as an integrated system, composed by subsystems, or modules, and defines the architecture that will consequently be used as reference for implementation.

The third Section describes the proposal and implementation of the general user interface for the system, specifically describing the main screen.

The fourth, fifth, sixth, seventh, eighth, ninth and tenth Sections describe in detail the proposal and implementation of each subsystem defined in the second Section, as well as the corresponding user interfaces.

3.2. Architecture

3.2.1 Module architecture

As can be perceived by the use of different programs, the core functionality of many steps in the photogrammetry process is self-contained. As such, it is natural for the proposed system to be decomposed to smaller subsystems, or modules, that will then interact with each other. Because the exact photogrammetry process may slightly differ according to the context and purpose, it is best for these modules to be as independent and modular as possible. Thus, seven different modules were identified and are listed below:

- **Image Input/Output (I/O) module**, which can be further decomposed to the following sub-modules:
    - **Metadata module**, responsible for extracting metadata information from an input image and also copy metadata to an output image;
    - **Image reader/writer module**, responsible for reading the image file and map it to an object with the pixel information to display it. It is also responsible for writing the image to a file;

- **Camera calibration module**, responsible for generating a calibration model of a camera based on photographs of calibrating points;
- **Lens distortion correction module**, responsible for correcting lens distortion present in photographs, using the model resulting from the calibration process;
- **Measuring module**, responsible for providing the different ways a user can measure and handling relevant measuring calculations, such as line length and angles between lines;
- **Unit conversion module**, responsible for handling unit conversion calculations;
- **Mathematical expressions module**, responsible for interpreting input expressions, variables and given values to output a value. Used, for example, in the context of allometric expressions;
- **Session module,** responsible for handling the different ways the user can save session information within the software to a file.

The different modules and the way they interact to fully realize the photogrammetry process is represented in Figure 10 and is described as follows (optional relationships between modules are represented with a dotted line):

1. User provided images are read by the Image I/O module. The image is both displayed, and its metadata information is extracted;
2. If relevant for the user, distortion present in input images is corrected by the Lens distortion correction module. This module uses a camera matrix model generated by the Camera Calibration Module to create new undistorted images, while preserving the original images;
3. Relevant measurements based on user input are made with the Measuring module, along with relevant results of mathematical expressions (e.g., allometric expressions) with the Mathematical expressions module. Mathematical expressions may optionally use measurements for its calculations;
4. If relevant for the user, measurements may be converted from pixel units to another unit system, such as metric. This conversion may optionally use metadata information for its calculations;

5. Measurements and mathematical expressions, denominated as layers, as well as optional metadata information for each input image are outputted into a file, called a session file.

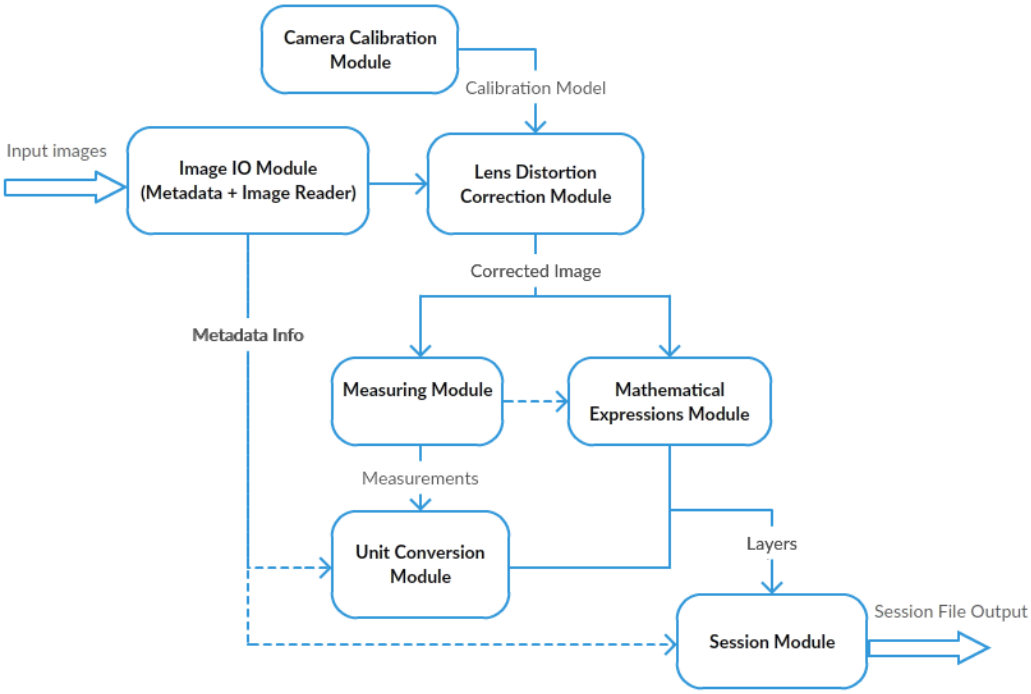


Figure 10 – Overview of module interaction.

3.2.2 Presentation software architecture

In practice, when implementing those modules, they should only contain logic respective to them, as to maintain their independence and modularity. Views that the user will see and interact should be implemented separately, allowing easy changes to the interface and decoupling responsibilities. This idea of separation of concerns in user interfaces are the basis of several software architectural patterns (Maxwell, 2017), hereon referred as 'presentation software architectures'.

MVC (Model-View-Controller) is a typical example of a presentation software architecture that was chosen to be implemented. More recently, it has been argued that this architecture is not modular and flexible enough, with other popular architectures being preferred, such as MVP (Model-View-Presenter) or MVVM (Model-View-ViewModel) (Maxwell, 2017). However, MVC is still generally easier to adopt, without as much overhead in implementation and was deemed sufficient for this use-case.

An overview of MVC architecture applied to this case can be seen in Figure 11. The view is responsible solely for displaying information and recording user input. The model defines the view data structure to be displayed. A controller class is used to handle user input, handle the model, delegate work to a relevant module and sometimes update the view directly. In a typical scenario, the controller would offload relevant work to a module and attribute the returning data to a view model. The model, consequently, has data binding capabilities that would trigger the view to be updated.

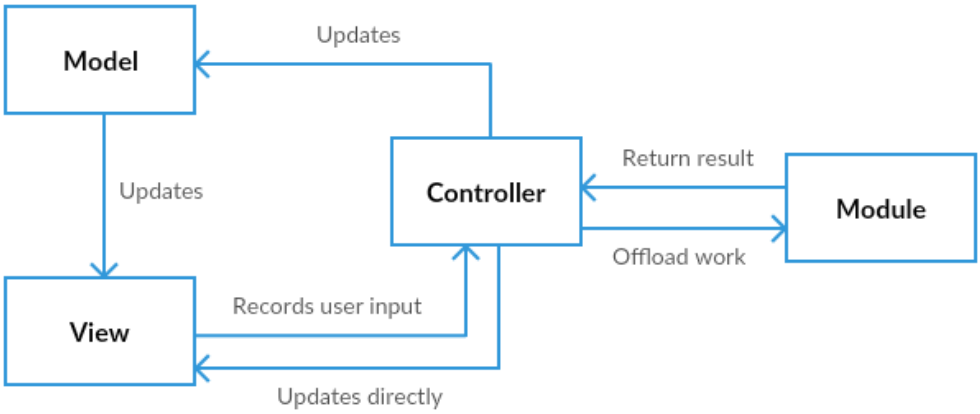


Figure 11 - Overview of the adopted MVC related architecture.

3.3. General user interface

The proposed architecture was implemented in Java for its cross-platform capabilities, and the user interface was implemented with the JavaFX framework, instead of the older Swing toolkit, along with the JFoenix library ("JFoenix," 2015) for an up-to-date Material design look and feel. The main objective in respect to user interface was to provide the maximum amount of information at a time, while retaining familiarity and simplicity to the user. As such, the main screen was divided into three main resizable panels along with a menu bar (as seen in Figure 12), described as follows:

- The **menu bar**, which is composed of several utilities, namely:
    - Opening and saving user sessions;
    - Importing images;
    - Exporting the session data as CSV (comma-separated values) file (refer to Section 3.10.3);
    - Camera calibration access and image distortion correction;

- Unit conversion options;
- View options, such as line display preferences;

- The **navigation panel**, where all the images imported in to a work session are listed. A thumbnail is shown beside the image name and image location;
- The **work panel**, which is the main work area, where most measuring operations over the image will be conducted. It has a tool bar on top, where the different available operations can be accessed, as well as image zoom and pan. In order to be intuitive, the layout and icons are similar to those in other image editor software;
- The **info panel**, where information about the image and the result of measuring operations can be observed and managed. It also is comprised of two tabs:
  - The **metadata tab**, where all metadata contained within the selected image is presented;
  - The **layers tab**, where information about the measurements and the result of mathematical expressions are presented. It has a tool bar at the bottom to create and delete layers.

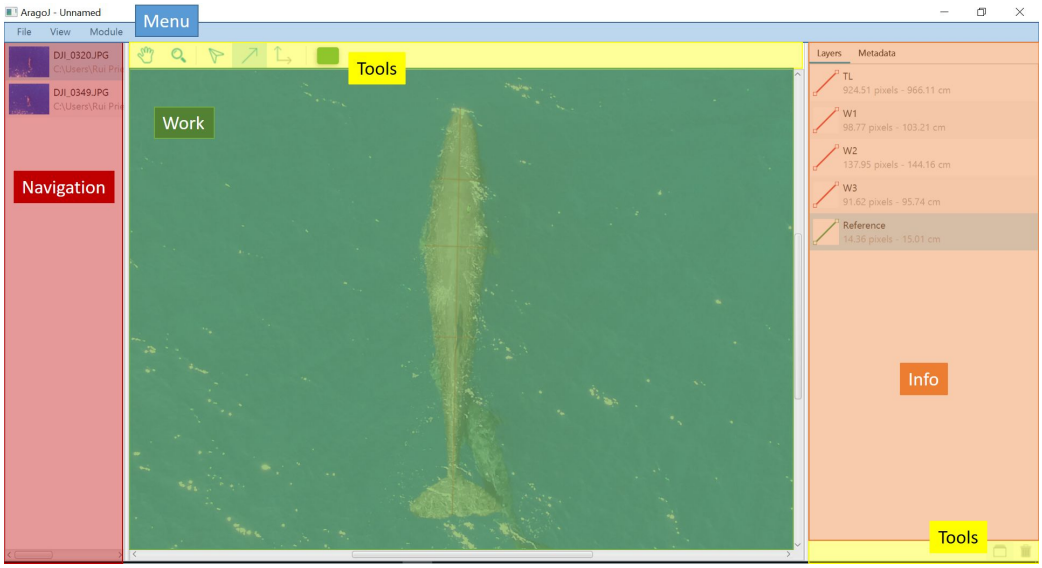


Figure 12 – The different components of the main screen represented by different colors.

Some modules described in the following Sections have specific views associated with it even though they are architecturally separated in terms of presentation, such as the calibration module. For context, these views are described in the respective module Section.

3.4. Image I/O module

The Image I/O module is responsible for reading and writing images, including pixel and metadata information.

The metadata submodule uses two different libraries: metadata-extractor (Noakes, 2002), to extract metadata information and present it to the user and Apache Commons Imaging library ("Commons Imaging: a Pure-Java Image Library," 2017) to copy metadata from one image to another, which is needed if an image is altered and metadata needs to be maintained when saving into an image file.

The image reader/writer submodule uses native Java and JavaFX functions to both read image files to display them, as well as write altered images back to files. Currently, the only format that is truly supported is JPEG, since it is the only format that is supported across all the functions and libraries used. It would be ideal, however, to further support TIFF and raw formats (or the standardized DNG).

3.5. Camera calibration module

3.5.1. Architecture

Even though there is separation of concerns throughout the program architecture in presentation logic, particular attention was paid to the calibration module. The program uses planar based calibration techniques provided by the Calib3d module from OpenCV. Throughout literature, it is clear that OpenCV is generally used as a benchmark for novel calibration techniques, and so the program should be flexible enough to allow the use of different techniques and algorithms without making many changes to the code. This flexibility relies on a layer of abstraction in order to generalize the calibration process and the overview of the architecture can be seen in Figure 13.

The calibration module is responsible for executing the calibration algorithms. The only public entry point to the module is the CalibrationManager class. This class is responsible for starting and stopping the process asynchronously, managing/executing the different algorithms and calling progress callbacks through the provided CalibrationRunListener. Depending on user configuration, the CalibrationManager can use different algorithms on different steps. The different algorithms are also generalized by the Calibrator and PointFinder interfaces.

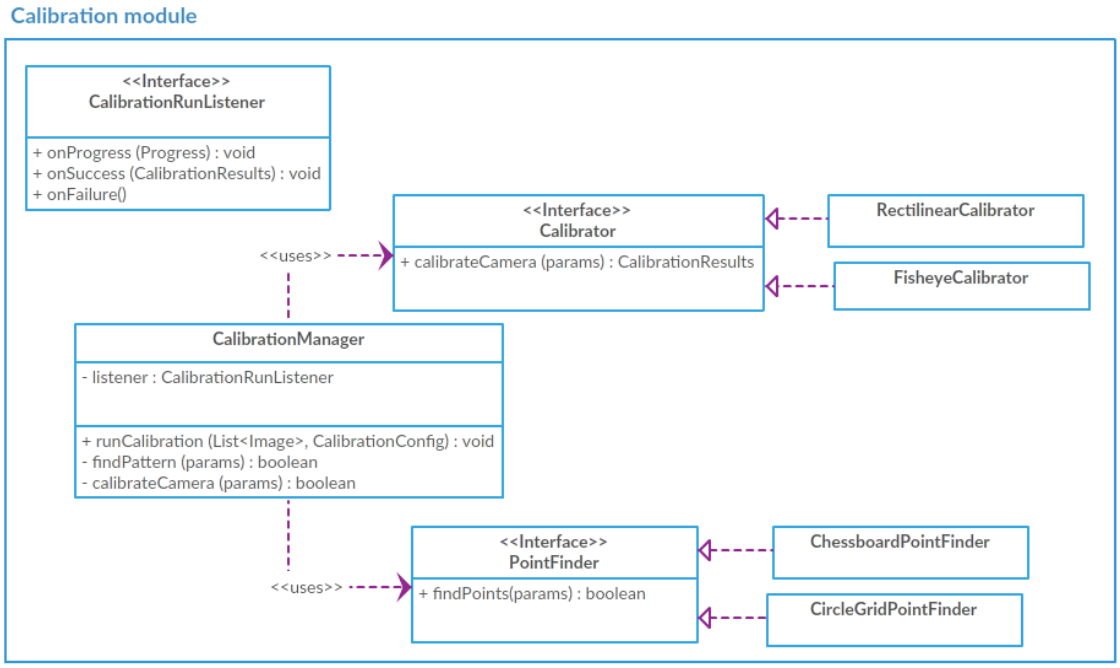


Figure 13 – Overview of the calibration module architecture.

3.5.2. Associated views

The calibration module, due to its complexity, has an entire screen and related dialogs dedicated to it. It shares a similar structure to the main screen described in Section 3.3. It is composed of the following components:

- The **menu bar**, which is composed of several utilities, namely:
    - Saving and exporting the calibration model. Saving and exporting are equivalent, but when exporting the user can choose where to export. If the user chooses to save, it saves automatically to a local directory for reasons described in Section 3.6;
    - Import calibration images;
    - Run calibration process;
    - Configure calibration process
- The **navigation panel**, where all the imported calibration images are listed. A thumbnail is shown beside the image name and image location. After calibrating, a status bar next to the thumbnail will appear either green if a calibration pattern was found for that image or red otherwise;

- The **image viewer panel**, which is similar to the work panel described in Section 3.3. It has a top toolbar where the user is able to zoom and pan the selected image as well as run and configure the calibration. It also displays the current configuration on the right. After calibrating, the calibration pattern is overlaid in each of the images that was used in the calibration, but not on those for which the pattern was not found.
- **The calibration output panel**, where information about the results of the calibration process is shown, including elapsed time, extracted camera model name, number of images with pattern detected and overall reprojection error, which is the arithmetic mean of each image reprojection error, see equation (4).

The delimited calibration screen before running the calibration process can be seen in Figure 14. The calibration progress and result views can be seen in Appendix A.

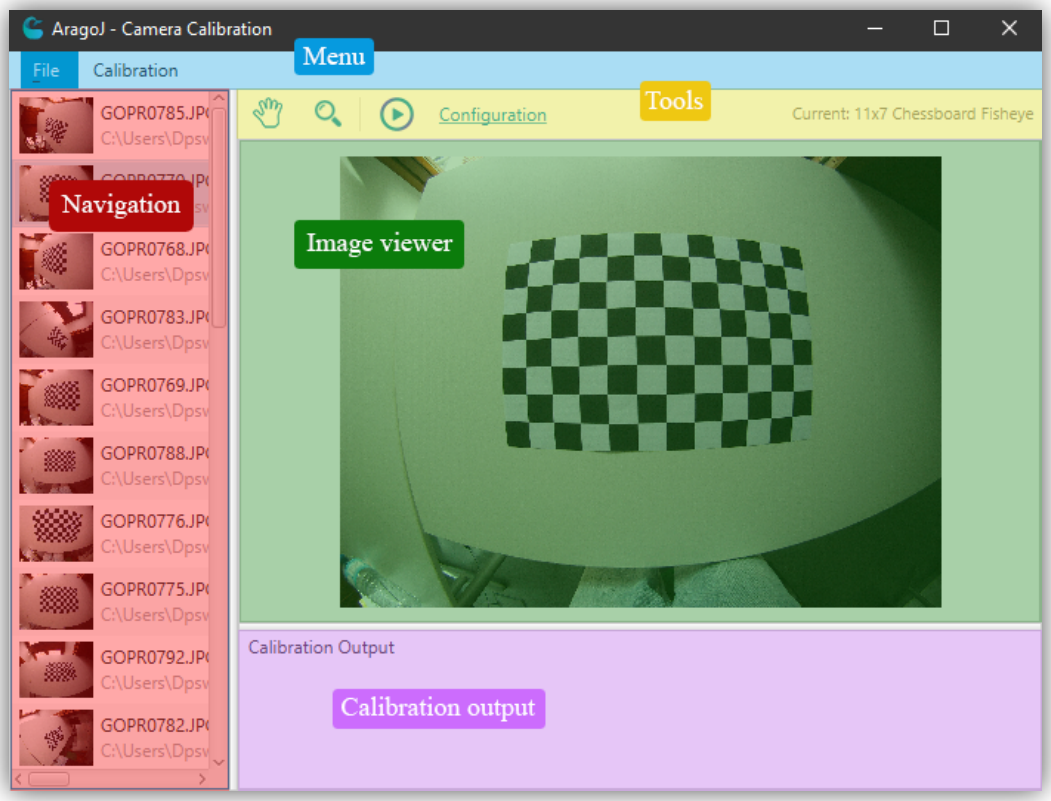


Figure 14 – The different delimited components of the calibration screen.

The calibration configuration view can be seen in Figure 15 and each option is explained in Section 3.5.3.

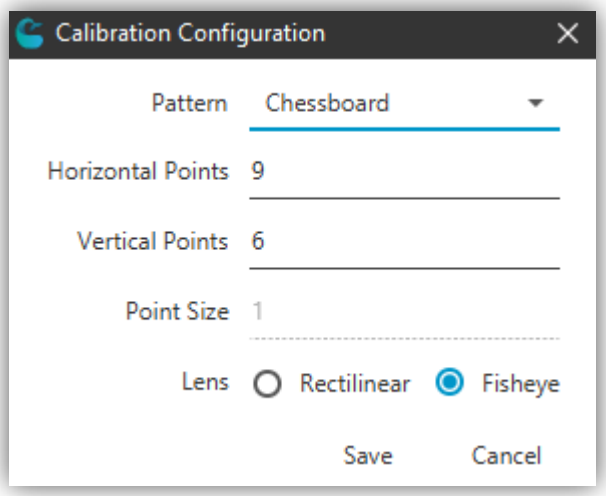


Figure 15 – Calibration configuration dialog.

3.5.3. OpenCV Calib3d – Implementation and limitations

As mentioned, OpenCV functions (specifically from Calib3d module) were used for both pattern detection and calibration. Calib3d allows a large amount of configuration when generating a calibration model. The most relevant variables were picked to be configurable by the user, as seen in Figure 15, and are listed as follows:

- **Pattern**: The calibration pattern that is present in the calibration images. It can either be a chessboard or a circle grid pattern;

- **Horizontal Points**: The amount of calibration points in the x axis. In the case of chessboard, it is the number of chessboard corners in a row. In the case of a circle grid it is the number of circles in a row;

- **Vertical Points**: The amount of calibration points in the y axis. In the case of chessboard, it is the number of chessboard corners in a column. In the case of a circle grid it is the number of circles in a column;

- **Point size**: Only applies to the circle grid pattern and is the rough size of the circles in millimeters. It relates to the way the blob detector, used to detect circles, works;

- **Lens**: The type of lens, either rectilinear or fisheye, which is associated with different calibration models.

OpenCV is written in C++ while the software developed in the scope of this work is written in Java. As such, to use OpenCV, a wrapper had to be used. The wrapper creates bindings so that it is possible to call C++ functions within Java. For this, there are two

components: a Java wrapper that defines the function interfaces that can be called, and a native library (for Windows a .dll file, OSX a .dylib file, Linux a .so file) that contains the C++ implementation of those functions. These two components communicate through Java Native Interface (JNI).

OpenCV does provide their own Java wrapper, however it has some shortcomings. As of version 3.4.0, their Java distribution has around 43.8 MB for x64 platforms and 27.4 MB for x86 platforms. This size is significant and hinders the distribution of the software, which by its own rounds to about 6 MB of size. Additionally, the wrapper is incomplete. For example, for the circle grid calibration pattern, it lacks support of blob detector which is crucial to identify the circle grid in different image resolutions

As such, a custom Java wrapper for OpenCV was created. It allowed to not only reduce size (~7 MB of size for x64, about ~84% size reduction) and access previously unported classes, but to also create custom routines/algorithms and facilitate the process of improving existing algorithms in the future.

3.5.4. Calibration file specification (.acalib)

In the program, a file extension, '.acalib', is used to distinguish files that contain the calibration model outputted from our program. It is a simple XML plain-text file that contains specific tag names related to relevant calibration information.

As per the XML specification, the file includes a prolog, declaring the XML version, encoding and standalone declaration although this is not necessarily needed. Every tag name is lowercase and uses underscores to separate words. The root element must be named 'calibration'. Every other element is a child to the root element and is defined below:

- **<name>**: Describes the name of the calibration model. This will be the name used throughout the program to reference the model and not the file name.
- **<lens>**: Describes the type of lens that this model describes. Currently only supports either "Rectilinear" or "Fisheye" values. These values directly affect how the <distortion_coefficients> element will be interpreted.
- **<intrinsic>**: Describes the 3×3 intrinsic matrix. Every value is formatted using the default Java double value notation, which uses standard scientific notation (with

E-notation) and a dot as decimal separator. Values are separated by spaces. When mapping to a 3×3 matrix, the values mapped from left to right, top to bottom.

- **<distortion_coefficients>**: Describes the distortion coefficients row vector with N elements. Every value is also formatted with the default Java double value notation. Values are separated by spaces. The number of elements is directly influenced by the lens type. For rectilinear it is 5 values (k1, k2, p1, p2, k3, in that order), for fisheye it is 4 values (k1, k2, k3, k4, in that order). See used equations in (4) for reference.

A full example of the file structure describing the calibration model of a fisheye lens can be seen in Appendix B.

3.6. Lens distortion correction module

The lens distortion correction module is responsible for correcting lens distortion using a provided calibration model. The module also uses OpenCV functions, namely undistort() from the Imgproc module and undistortImage() from the fisheye namespace. The first function is used in the case of rectilinear lenses and the second in the case of fisheye lenses, which has the target image and the calibration model as inputs, and the corrected image as output.

The dialog associated with the module can be seen in Figure 16. As it launches, it automatically checks a relative directory to the program location called ‘calibs’ for any ‘.acalib’ files and automatically imports them. The user can also manually locate and import them if the calibration model files are in other locations. The ‘Apply to all’ checkbox simply applies the undistortion to every image in the session if checked, or to only the selected image if unchecked. Undistorted images are saved with a suffix (‘_u’) relative to the original files’ name, and the original images are preserved.

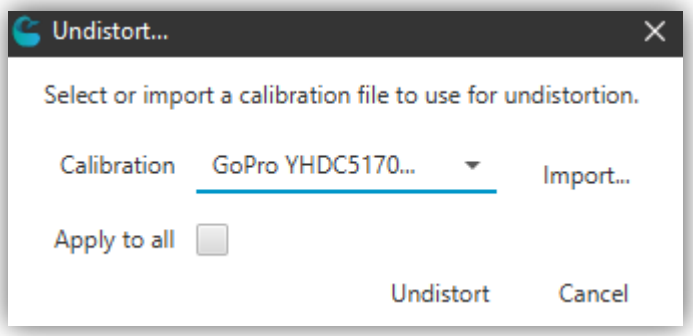


Figure 16 - Undistort dialog.

3.7. Measuring module

The measuring module is responsible for measuring distance between two points (lines) and angles between lines. It is associated with two main features:

- **Line measurement**, where the user is able to overlay a line defined by two adjustable points and adjustable line color. The line is remeasured every time any of the two points are adjusted. This is easily done applying the distance formula;
- **Angled line constraint**, where the user is able to overlay a line in an angle relative to an existing line. The constraint is applied in two general steps:
    1. Calculate the length and direction of the line between the anchor point, A, and the cursor point, to define the vector v;
    2. Given that θ is the input angle, the cursor point x and y coordinates are remapped to x' and y' respectively, using the following functions.

$$x' = Ax + v * \cos\theta$$

$$y' = Ay + v * \sin\theta$$

The resulting lines information are displayed in the layers section as can be seen in Figure 17.

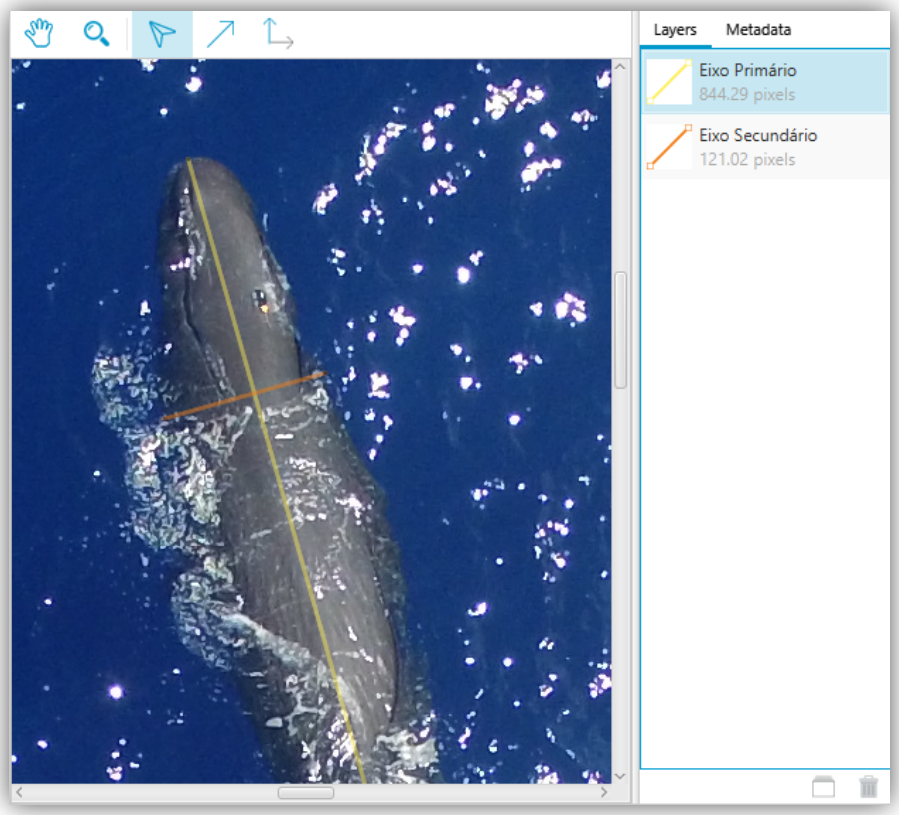


Figure 17 – Example of two perpendicular measuring lines.

3.8. Unit conversion module

The unit conversion module is responsible for converting pixel measurements to another desired unit. It handles both reference-based conversion and distance-based conversion, as explained in Section 2.5.

Reference based conversion is associated with the dialog represented in Figure 18. It is based on a known reference measurement, S, on the photograph and can be very easily calculated using (6), for example, in the case of centimeter conversion. M is the measurement taken by the user, the "Length in pixels" field corresponds to S_{px} and the "True length" to S_{cm}. The scaling ratio is simply S_{cm} divided by S_{px} and it is multiplied by every relevant pixel measurement. "Apply to all" simply refers if the conversion is to be applied to every session image measurements or only the current one.

$$M_{cm} = \left(\frac{M_{px} S_{cm}}{S_{px}}\right) \tag{6}$$

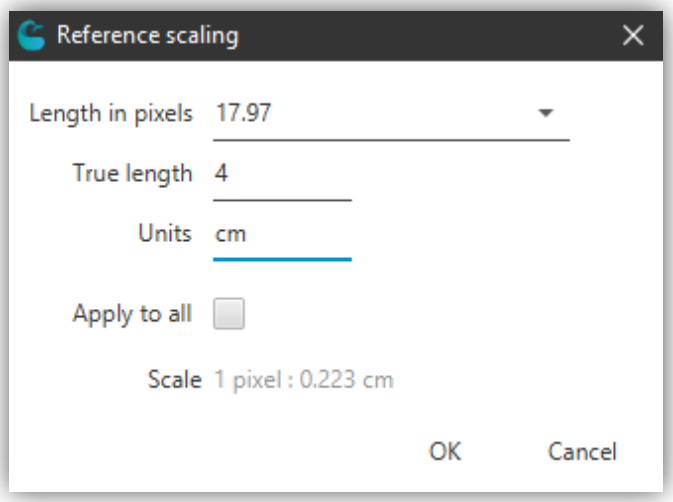


Figure 18 – Reference scaling dialog example.

Distance-based conversion is associated with the dialog represented in Figure 19. It uses the pixel dimension, the focal length and the distance from the focal plane to the object to apply the conversion as described in (7). For simplicity, the process is done using the standardized metric system explicit in the dialog, but can later be converted to other units. Preset refers to existing information about some cameras sensor size in a local database.

The first step in the calculation is finding the pixel dimension. Pixel dimension is sometimes given by the manufacturer, but more often the camera sensor dimensions are more easily found. As such, the real pixel size can be found by dividing the camera sensor width, Sw, and height by the image pixel width, imW, and height respectively. Since the pixels are generally square, only the width or the height scaling ratio is needed. If the pixel size at the focal length, Fl, is known, extrapolating to the object distance is now trivial. The object distance is represented in (7) by $(H + \beta)$ because it is often used in the context of drones. Using drones, the distance is the altitude of the drone, H, which can be extracted via image metadata, with an added correction factor, β. If the context is, for example, at sea level, the correction factor corresponds to the boat height above sea level, since the drone is initially calibrated at the boat height and not sea level. If no correction is necessary, the user can either leave the field empty or set β to zero.

$$M_{cm} = M_{px}\left(\frac{Sw * (H + \beta) * 100}{Fl * imW}\right) \tag{7}$$

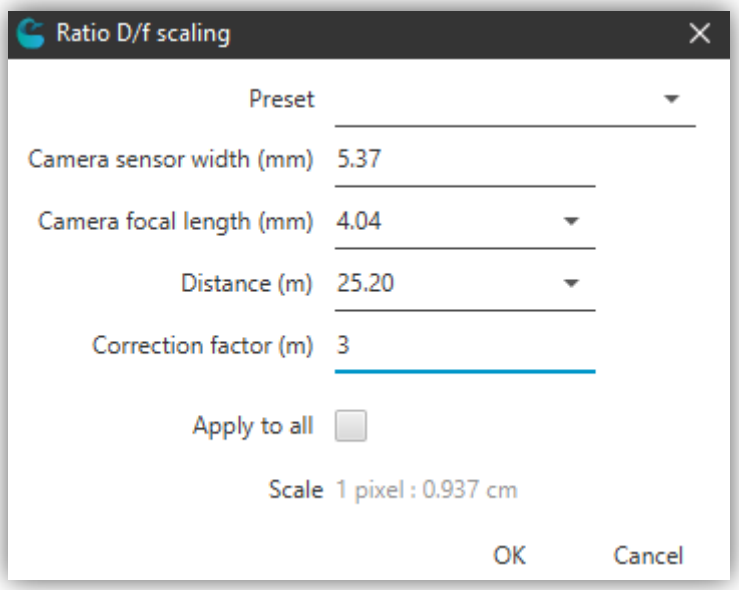


Figure 19 – Distance scaling dialog example.

After the conversion/scaling is applied, the target unit length is presented next to the pixel length as shown in Figure 20.

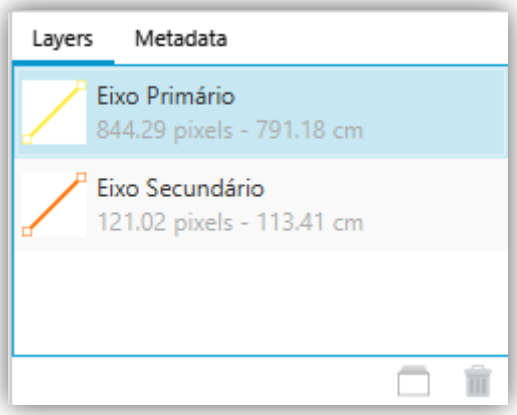


Figure 20 – Example of resulting centimeter conversion.

3.9. Mathematical expressions module

3.9.1 Description and associated views

The mathematical expressions module is responsible for handling input mathematical expressions with variables and computing the value of the expression given the variables values. It is also responsible for storing expressions into files and loading expressions from files. This is a generalized solution to the problem of allometric expressions and is suited for any use-case that requires reusable mathematical expressions.

The module is associated with the dialog represented in Figure 21. The user is able to create and remove mathematical expressions, as well as name them. Mathematical expressions are strings inputted by the user and a library named mXparser (Gromada, M., n.d.), which is used to validate and interpret the string automatically after the user stops writing in the expression field. The library is also able to detect possible variables within the expression which are automatically added to a table below, where the user is then able to set its value. The value can be set to any numerical value, and a convenient drop-down list is also present to easily access the layers' values in both the pixel and the scaling unit. Finally, as the user presses the OK button, the resulting value is added to the layer list, as can be seen in Figure 22.

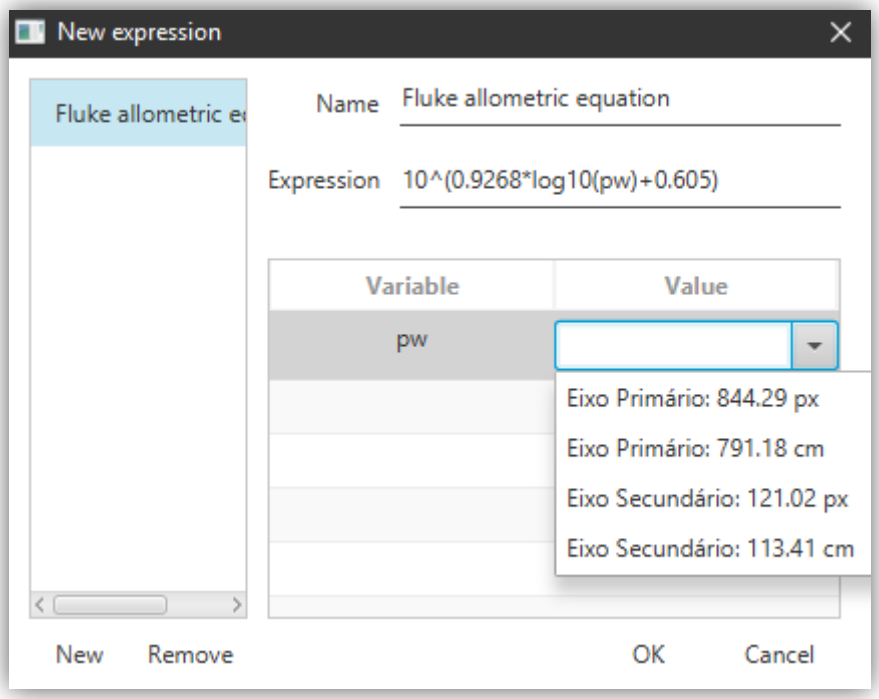


Figure 21 – Example of setting mathematical expression and attributing variable value.

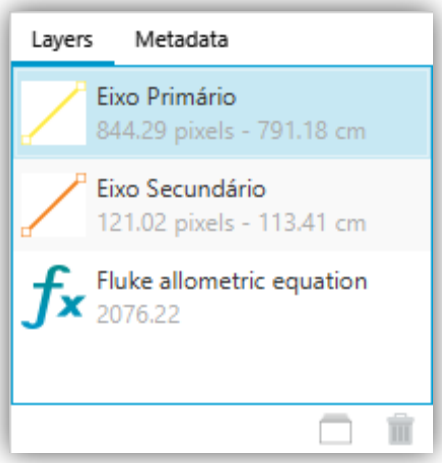


Figure 22 – Example of a resulting expression value in the layer list.

.

3.9.2 Expression file specification (.xml)

As the user continuously fills the name and expression fields, it automatically saves it to a file, called 'expressions.xml' under a directory in the root executable location called 'data'. This file and the containing expressions are loaded when the dialog represented in Figure 21 is shown. It follows the same XML standard as the '.acalib' file but has no custom file extension name as it is used and managed internally by the program.

The root element is named 'expressions'. For each mathematical expression, a parent element named 'item' is created containing two child elements:

- **<name>**: Describes the name of the mathematical expression;
- **<expression>**: Expression string to be interpreted by the program.

Note that variable values are not stored, as only the expressions are meant to be reusable. An example of the file structure describing one mathematical expression can be seen in Appendix C.

3.10. Session module

3.10.1 Description

The session module is responsible for handling the input and output of sessions from files, as well as manage sessions throughout the lifecycle of the application. A session is akin to a project in many other image editing software (e.g., .PSD file for Adobe

PhotoShop), and in this context refers to all related work done by the user that can be saved to a file. This module specifically handles two use-cases:

1. Opening and saving session information through a '.axml' file. This file only encodes necessary information to be readable by the program and is not meant to be read by the user. It is henceforth referred as 'program-specific session file';
2. Exporting session information to a .CSV file. This file is meant to be directly read, analyzed and used by the user. It is henceforth referred as 'user-specific session file'.

3.10.2 Program-specific session file specification (.axml)

A program-specific session holds information about each imported image, such as file path, zoom information and layer information. It follows the same XML standard as previous file specifications but has '.axml' as the file extension name for easily locating and opening sessions.

The root element is named 'session'. For each imported image, a parent element named 'image' is created containing three child elements:

- **<layers>**: Contains information about the created images' layers;
- **<sourceImagePath>**: The absolute image path;
- **<zoom>**: Describes information about the zoom settings that the user set when working in the image.

In the **<layers>** tag, for each layer there's a child element describing the type of layer, either 'line' if it is a measuring line or 'expression' if it is a mathematical expression. The line layer contains the following child elements:

- **<name>**: Name of the layer;
- **<startPointX>**: The X axis coordinates of the line's starting point;
- **<startPointY>**: The Y axis coordinates of the line's starting point;
- **<endPointX>**: The X axis coordinates of the line's ending point;
- **<endPointY>**: The Y axis coordinates of the line's ending point;
- <**color>**: The color of the line.

The expression layer contains the following child elements:

- **<name>**: Name of the mathematical expression;
- **<expression>**: Expression string to be interpreted by the program;
- **<variable> (for each variable)**: Contains two child elements describing information about a variable, namely:
    - **<name>**: Name of the variable;
    - **<value>**: Value of the variable.

In the **<zoom>** tag, there are three additional child elements, namely:

- **<scale>**: The scale ratio of the image as observed in the work panel;
- **<hValue>**: The horizontal scrolling location of the zoomed in image;
- **<vValue>**: The vertical scrolling location of the zoomed in image.

An example of the file structure describing a program-specific session with all these described attributes can be seen in Appendix D.

3.10.3 User-specific session file specification (.CSV)

A user-specific session holds information about each image source path, selected metadata information and layer information, where the values are delimited by commas (comma-separated values, or CSV). CSV was chosen not only because it is easy to write to, but because it is also easy for the user to interpret and analyze. It is structured as follows.

1. The first line is the headers line. It describes each column value and is comma-separated. Naming is structured as follows:
    a. First value is always “Source” and refers to the absolute image path;
    b. The following values are selected image metadata tag names by the user, but are not required. Metadata selection for exporting is done as seen in Figure 23;
    c. After the optional metadata tag names, the layers names are inserted followed by the unit name in parentheses (e.g., ‘Layer1 (px)’). Layers names cannot be repeated, except for different unit names. If two different images contain a layer with the same name, their corresponding values will be placed in the same column;

2. For each image, a line is inserted with the corresponding values of the columns defined in the first line.

An example of the file structure describing a user-specific session with all these described attributes can be seen in Appendix E.

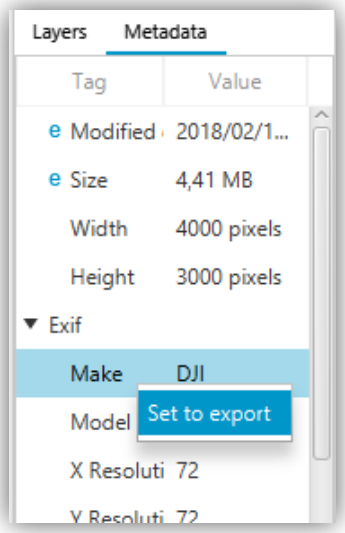


Figure 23 – Setting a metadata tag to be exported. If set to export, the tag will have a blue 'e' preceding it. Whole metadata directories, such as Exif can be set to export.

Chapter 4 – Results analysis and discussion

4.1. Introduction

This chapter is composed of four Sections and analyzes the usability and processing time of the proposed solution compared with the solution described in Section 2.2.2. The first Section introduces and describes the chapter structure.

The second Section defines the specific questions that will be analyzed and studied, as well as details about the experiment and about how data was collected to answer them.

The third Section explores the quantitative and qualitative results of the experiment defined in the second Section and analyzes it critically

The fourth Section analyzes the camera calibration module independently, which was not evaluated in the previous Sections, by comparing user interfaces with a competing software.

4.2. Data collection

In this chapter, two main questions are studied through an empirical study, namely:

1. Does the use of the proposed software generate faster, and equivalent or better results than previous solutions?
2. Is the proposed software more intuitive, easier and convenient to operate, compared to previous solutions?

To answer these questions, a qualitative and a limited quantitative approach was used. A quantitative approach was limited by only one participant for two main related reasons:

1. Outside specific research groups, the number of people that understand the basic underlying concept and have a use for the full proposed solution are somewhat limited, and hard to reach;

2. As with any other program, the proficiency in the use of the program developed in the scope of this work can only be attained through a learning curve. At the time of the completion of this work only one person was considered proficient in the use of the program and thus the only who could give unbiased results when comparing with other solutions.

To answer the first question, "*Does the use of the proposed software generate faster, and equivalent or better results than previous solutions?*", an experiment was conducted

where the researcher was asked to complete the photogrammetry process detailed in Section 2.2.2, and then the equivalent process with the software developed in the scope of this work. To answer the second question, "*Is the proposed software more intuitive, easier and convenient to operate, compared to previous solutions?*", an informal interview was conducted afterwards asking to compare both processes qualitatively. Camera calibration was the only step evaluated separately and differently, as will be described in Section 4.4.

A set of ten aerial pictures of sperm whales (*Physeter macrocephalus*) was used for the experiment. All pictures were obtained during fieldwork by marine mammal researchers of the Institute of Marine Research at Azores, under permit from the Azores Regional Government. The pictures were captured using a DJI Phantom 3 Standard quadcopter, hereby referred as 'drone'.

Height above sea-level was estimated by the absolute air pressure readings from the drone's inbuilt barometer (Durban et al. 2015), using a correction factor of 0.4 meters due to the height above sea-level of the take-off platform on the boat used to deploy the drone. Barometric readings, along with other relevant information such as the drone geographic location from an inbuilt GPS, camera orientation, among other, are written as Exif and XMP metadata.

The experiment involved extracting four measurements of the animal body from each picture, in pixel units, merging those measurements with relevant extracted metadata in the respective picture and converting measurements to metric units. The measurements taken consisted in the animal length and three animal widths at different points along the animal length, perpendicular to the length axis (Figure 24). For the solution referenced in Section 2.2.2, GIMP was used to retrieve measurements.

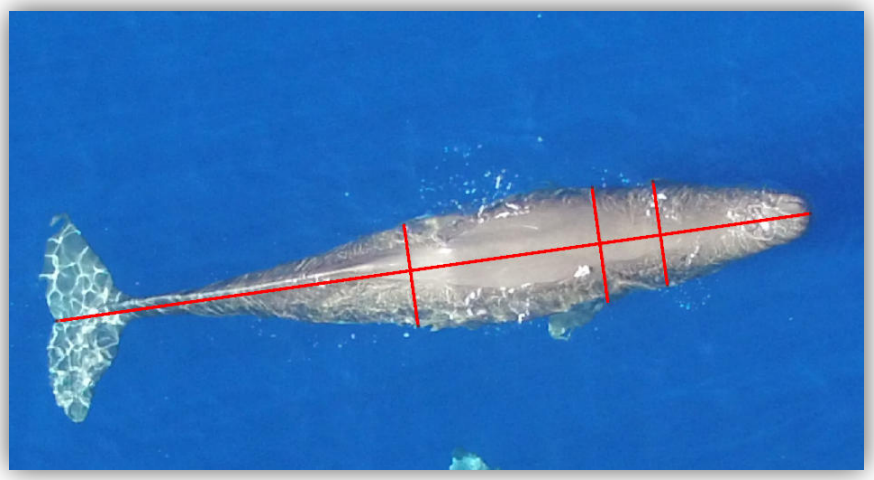


Figure 24 – Experiment extracted measurements: one for animal length, three for width.

As mentioned above, the participant is an experienced user and is familiar with both protocols (the current and the new proposed solution). Processing time for both solutions was recorded using a digital chronometer and two main tasks were considered:

1. Time for acquiring measurements;
2. Time to scale and export results;

The experiment was repeated three times for each protocol and resulting times were averaged.

4.3. Overall results

4.3.1 Quantitative time results

The results for processing time are presented in Table 1. Time for acquiring measurements decreased by ≈40%, from 31 minutes and 43 seconds to 19 minutes and 1 second. Processing time for conversion/scaling and exporting results also decreased from 3 minutes and 25 seconds to 9 seconds (≈95% decrease). Overall the total time decreased from 35 minutes and 8 seconds to 19 minutes and 10 seconds, a decrease of ≈45,4%.

The biggest factor in decreasing the total time is the time for acquiring measurements. While the method for measuring lines is similar in both solutions, there is a key difference: in the current solution, measurements are taken and annotated externally one at a time, while the proposed solution allows the management of multiple measurements.

The processing time for conversion and exporting is also significantly faster since the proposed solution natively handles this problem, while with the current solution the document structure and input values (and headers) for each image need to be inserted manually. This also includes relevant metadata information and conversion ratios.

Table 1 - Quantitative results of time taken to perform two different steps of the photogrammetry process.

	Time for acquiring measurements (minutes:seconds)	**Time to scale measurements and export results (minutes:seconds)**	**Total time (minutes:seconds)**
Using non-dedicated software	31:43	03:25	35:08
Using proposed software	19:01	00:09	19:10

4.3.2 Qualitative interview results

The most noted difference between both solutions was the ease of use and the workflow itself. Being able to complete the whole photogrammetry process in one program and not having to switch between programs also avoids possible data corruption and overall streamlines the process. The three resizable view panels were also noted ideal in visualizing all the necessary information simultaneously.

A few quality of life details in the proposed solution were also noted, such as remembering the last directory used when saving/opening files, having the option to apply the same scaling to every image, having suggested values when performing unit conversion, having a thumbnail and source path for each image on the navigation panel and being able to set entire metadata groups, such as Exif, to be exported.

However, there were also a few notes and suggestions to improve in some regards, namely:

- When creating a line, the zoom does not scale towards the mouse pointer location which results in the line having to be readjusted later;

- When setting scale/unit conversion using distance conversion for each image, some fields needs to be inputted repeatedly, such as camera sensor width and focal length, where they could be pre-filled;
- If the image size is big enough, when zooming out the image, sometimes the measurements can be almost impossible to visualize because the measurement lines have a fixed thickness.

Further discussions and suggestions can be accessed in the Github platform (https://github.com/franciscoaleixo/AragoJ/issues). Notice, however, that overall, the proposed solution was recognized as a significantly better solution in terms of experience, usability and productivity.

4.4. Camera calibration

Since the proposed solution uses OpenCV for camera calibration without any inner code changes, it also means that it performs similarly to other software that also uses OpenCV, including Calib3V (Balletti et al., 2014). As such, user experience is the biggest difference and most relevant point to compare.

To compare both solutions, each step in the calibration process was analyzed, namely:

1. **Importing images**: The proposed solution uses a native dialog for finding images and importing them, which makes finding the correct files very easy while Calib3V does not. Furthermore, imported images can be seen in the navigation panel in the proposed solution, while there is no indication of what the imported images on the Calib3V are;
2. **Configuring the calibration process:** Calib3V only allows the calibration of circle grid patterns (Annex A), while the proposed solution also allows, additionally, the chessboard pattern. Horizontal and vertical point number must be defined in both, but in Calib3V width and height of circles as well as target distance must also be defined by the user, while the proposed solution does not require it. Calib3V also features, for each image, a region selection ability, where the pattern detector can be tweaked;
3. **Calibration execution and results:** In Calib3V there is not much feedback to the user about the calibration execution and results, apart from the resulting camera

matrix, if it succeeds (Annex B). In the proposed solution, both the execution status and results are reported.

It is important to note that Calib3V is primarily a prototype, which means that many reported usability issues are to be expected. Further, additional configuration required in Calib3V might lead to better detection rates. In fact, OpenCV allows a vast amount of configuration, including allowing manual input for focal length, if known, for possibly better calibration models. While none of this is present in both solutions, it is interesting to further explore it in the future. Finally, Calib3V is also focused on action camera's calibration, while the proposed solution attempts to generalize the application scenarios, which does lead to the expected differences in configuration.

Chapter 5 – Conclusions and recommendations

5.1. Introduction

This chapter is composed of five Sections and derives conclusions from the present work. The first Section introduces and describes the chapter structure.

The second Section explores the main key points from the present work, including methodology, implementation and results.

The third Section describes possible contributions to the scientific community as a result of this work, and specifically as a result of the developed solution.

The fourth Section analyzes the key limitations in the study and results of the solution explored in Chapter 4.

The fifth Section suggests future improvements to both the solution and the analytical study and evaluation of the solution.

5.2. Main conclusions

The investigation objectives of formulating and implementing an easier, faster and free method of executing the photogrammetry process in the biological context were generally achieved. The module architecture and presentation architecture proved to simplify implementation and allowed better maintenance of the software. Additionally, it allowed to easily make changes that originated from the received feedback. By using Java in implementation, cross-platform was also made possible.

The agile Kanban-based methodology also worked very well, although it proved difficult to maintain a strict schedule for development and testing due to extraneous circumstances. Despite this, it led to constructive public discussions, that can be accessed on the Github platform (https://github.com/franciscoaleixo/AragoJ), on how the software behaved, and what could be improved, which also proved helpful in writing the present text.

However, there are also some open issues to improve in the proposed solution, including extending image file support beyond JPEG files, improving user experience in some edge cases, reducing dependency from third-party software libraries, improving and rethinking camera calibration configuration options, among others. There are also some additional features that would be interesting to be implemented, such as angle and area measuring tools that are further discussed in Section 5.5.

The quantitative and qualitative results gathered were also encouraging, where processing time was reduced approximately by 45%, and usability feedback was positive. However, the results were also somehow limited in number and further tests are necessary to reach sound conclusions. Since the solution is also proposed to be general, different processes and use-cases comparisons would also provide useful information and would probably lead to further suggestions for improvement.

Overall, the present work achieves at building a starting point in the unification and simplification of the different photogrammetry steps in a biological context in a single free software program. It consequently replaced the previous photogrammetry solution in use by the researchers at Azores Whale Lab (R. Prieto, personal communication, 2017). The developed software is named AragoJ and the releases are openly available in the Github platform (https://github.com/franciscoaleixo/AragoJ/releases). Currently, the source code is not openly available but there are plans in the future to make it available to the public in the same platform.

5.3. Contributions to the scientific community

As mentioned, the present work main focus was the integration and simplification of the different photogrammetry steps when measurements cannot be automatically taken. This manual process can be a very laborious process, especially if there are a large amount of measurements to be recorded, which the proposed solution has shown to alleviate.

As such, the present work contributions are mainly related with productivity and quality of life improvements over previous solutions in the context of photogrammetry. Fields in the area of biological research, which, in diverse cases, require manual close-range photogrammetry techniques directly benefit from the present work. A shown and proven example is the adoption of the proposed photogrammetry solution in the current and future studies of sperm whales by researchers at Azores Whale Lab (R. Prieto, personal communication, 2017), where the benefits of the proposed solution were well noted.

In a more general sense, the present work contributes to any field that makes use of photogrammetry techniques and require manual recording of measurements. Further, because the presented solution still respects the modularity of the different components within the photogrammetry process, it can also be used partially in other contexts. For

example, it can be used independently as a camera calibration tool, a measuring tool, or an image metadata viewer tool.

Finally, an important and differentiating aspect of the presented solution is that it is portable, free and supports cross-platform, which further means that it can be universally used by anyone with a desktop with virtually no further restriction.

5.4. Study limitations

Although the results presented in Chapter 4 were encouraging and help confirm the initial objective of increasing productiveness and intuitiveness, there were clear limitations present within the study.

The reduced sample size is a big limiting factor in accurately estimating how much faster the proposed solution is compared to other solutions. Additionally, the present work was only compared to the Azores Whale Lab photogrammetry solution, and even then, the use case of boat photoshoots was not compared, thus not every module was compared. Only the most common use case, with aerial photoshoots, was compared. Other photogrammetry processes, albeit generally similar, were not tested or compared.

Similarly, learning curve for both solutions was not explored/tested, which could also lead to interesting results. Since the presented solution is an integrated system capable of performing the whole photogrammetry process, it is assumed that it would be easier, in principle, to operate and learn, whereas the previous Azores Whale Lab solution relied on the user having to learn to operate different software tools. However, further tests with inexperienced users are needed in order to confirm that assumption.

Another present limiting factor is possible bias in the qualitative results. Researchers at Azores Whale Lab were involved over the course of the solution development, namely in providing feedback. As such, the solution was, to a certain extent, tailored according to their feedback. It is then natural that the final evaluated solution received positive feedback throughout. While it is certainly reasonable to conclude that the solution is appropriate for their use cases, and better in terms of productiveness than their previous solution, there is not enough data to generalize and assume the same for other possible use cases.

5.5. Future work

Due to time constraints, many useful features, tweaks and experiments were not done. Future work concerns two main areas:

1. **Improvements to the solution**: While the present solution has already been shown to meet the original requirements and have considerable advantages to previous solutions, there are still areas for improvement, namely:
    a. Implementation of additional measuring tools to measure areas, angles and curved features (with segmented lines);
    b. Implementation of image processing tools to aid the measuring process, such as edge detection, greyscale filtering, and so on;
    c. Research and implementation of automatic or semi-automatic measuring tools to further improve productivity;
    d. Experimenting and reviewing novel camera calibration techniques to further improve the calibration parameters accuracy. Also tweaking the calibration configuration process and ideally simplify it while retaining good results;
    e. Implementing support for other image formats other than JPEG related formats.
2. **Further quantitative and qualitative analysis**: As noted in Section 5.4, there were notable limitations in the quantitative and qualitative results. Some suggestions for improvement in this regard include:
    a. Increasing the number of interviews for both quantitative and qualitative analysis;
    b. Usability and learning curve testing with individuals that were not involved in the process of the solution proposal and development;
    c. Increasing the number of use cases and contexts studied to fully evaluate the solution under different application scenarios.

Bibliography

Abeles, P. (2012). BoofCV. https://boofcv.org/. Retrieved from https://boofcv.org/index.php?title=Main_Page

Abràmoff, M. D., Magalhães, P. J., & Ram, S. J. (2004). Image processing with ImageJ. *Biophotonics international, 11*(7), 36-42.

Adams, D. C., Rohlf, F. J., & Slice, D. E. (2004). Geometric morphometrics: ten years of progress following the 'revolution'. *Italian Journal of Zoology, 71*(1), 5-16.

Adams, D. C., Rohlf, F. J., & Slice, D. E. (2013). A field comes of age: geometric morphometrics in the 21st century. [journal article]. *Hystrix - Italian Journal of Mammalogy, 24*(1), 7-14. doi: 10.4404/hystrix-24.1-6283

Adobe. (2012). XMP Specification Part 1 - Data Model, Serialization and Core Properties. https://wwwimages2.adobe.com/content/dam/acom/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2016-08/XMPSpecificationPart1.pdf.

Anonymous. (2015). GIMP-The GNU Image Manipulation Program.

Arago, F. (1839). *Rapport de M. Arago sur le daguerréotype, lu à la séance de la Chambre des députés, le 3 juillet 1839, et à l'Académie des sciences, séance du 19 août*. Paris: Bachelier (Paris).

Balletti, C., Guerra, F., Tsioukas, V., & Vernier, P. (2014). Calibration of action cameras for photogrammetric purposes. *Sensors, 14*(9), 17471-17490.

. Basic concepts of the homography explained with code. (n.d.) *https://docs.opencv.org/ref/master/d9/dab/tutorial_homography.html*.

Blackith, R. (1957). Polymorphism in some Australian locusts and grasshoppers. *Biometrics, 13*(2), 183-196.

Bouguet, J.-Y. (2000). Matlab camera calibration toolbox. *Caltech Technical Report*.

Boyde, A., & Ross, H. (1975). Photogrammetry and the scanning electron microscope. *The Photogrammetric Record, 8*(46), 408-408.

Bradski, G., & Kaehler, A. (2000). OpenCV. *Dr. Dobb's journal of software tools, 3*.

Breuer, T., Robbins, M. M., & Boesch, C. (2007). Using photogrammetry and color scoring to assess sexual dimorphism in wild western gorillas (*Gorilla gorilla*). *American Journal of Physical Anthropology, 134*(3), 369-382. doi: 10.1002/ajpa.20678

Burnett, J. D., Lemos, L., Barlow, D., Wing, M. G., Chandler, T., & Torres, L. G. (2018). Estimating morphometric attributes of baleen whales with photogrammetry from small UASs: A case study with blue and gray whales. *Marine Mammal Science*.

Chen, D., & Zhang, G. (2005). A new sub-pixel detector for x-corners in camera calibration targets.

Claus, D., & Fitzgibbon, A. W. (2005). *A rational function lens distortion model for general cameras.* Paper presented at the Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.

. Commons Imaging: a Pure-Java Image Library. (2017). https://commons.apache.org/proper/commons-imaging/.

Datta, A., Kim, J.-S., & Kanade, T. (2009). *Accurate camera calibration using iterative refinement of control points.* Paper presented at the Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference.

Dawson, S. M., Bowman, M. H., Leunissen, E., & Sirguey, P. (2017). Inexpensive aerial photogrammetry for studies of whales and large marine animals. *Frontiers in Marine Science, 4*, 366.

De la Escalera, A., & Armingol, J. M. (2010). Automatic chessboard detection for intrinsic and extrinsic camera parameter calibration. *Sensors, 10*(3), 2027-2044.

Deakos, M. H. (2010). Paired-laser photogrammetry as a simple and accurate system for measuring the body size of free-ranging manta rays Manta alfredi. *Aquatic Biology, 10*(1), 1-10. doi: 10.3354/ab00258

Donné, S., De Vylder, J., Goossens, B., & Philips, W. (2016). MATE: Machine learning for adaptive calibration template detection. *Sensors, 16*(11), 1858.

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization, 10*(2), 112-122.

Durban, J. W., Fearnbach, H., Barrett-Lennard, L. G., Perryman, W. L., & Leroi, D. J. (2015). Photogrammetry of killer whales using a small hexacopter launched at sea. *Journal of Unmanned Vehicle Systems, 3*(3), 131-135. doi: 10.1139/juvs-2015-0020

Durban, J. W., Moore, M. J., Chiang, G., Hickmott, L. S., Bocconcelli, A., Howes, G., . . . LeRoi, D. J. (2016). Photogrammetry of blue whales with an unmanned hexacopter. *Marine Mammal Science, 32*(4), 1510-1515. doi: 10.1111/mms.12328

Elewa, A. M. (2010). *Morphometrics for nonmorphometricians* (Vol. 124): Springer.

Faugeras, O. D., Luong, Q.-T., & Maybank, S. J. (1992). *Camera self-calibration: Theory and experiments.* Paper presented at the European conference on computer vision.

Fraser, C., & Brown, D. (1986). Industrial photogrammetry: New developments and recent applications. *The Photogrammetric Record, 12*(68), 197-217.

Fussell, A. (1982). Terrestrial photogrammetry in archaeology. *World Archaeology, 14*(2), 157-172.

Galbany, J., Stoinski, T. S., Abavandimwe, D., Breuer, T., Rutkowski, W., Batista, N. V., . . . McFarlin, S. C. (2016). Validation of two independent photogrammetric techniques for determining body measurements of gorillas. *American Journal of Primatology, 78*(4), 418-431. doi: 10.1002/ajp.22511

Gentleman, R. C., Carey, V. J., Bates, D. M., Bolstad, B., Dettling, M., Dudoit, S., . . . Gentry, J. (2004). Bioconductor: open software development for computational biology and bioinformatics. *Genome biology, 5*(10), R80.

Googe, W. D., Eichhorn, H., & Luckac, C. F. (1970). The overlap algorithm for the reduction of photographic star catalogues. *Monthly Notices of the Royal Astronomical Society, 150*, 35.

Grip, W. M., Grip, R. W., & Morrison, R. D. (2000). Application of aerial photography and photogrammetry in environmental forensic investigations. *Environmental Forensics, 1*(3), 121-129.

Ha, H., Perdoch, M., Alismail, H., Kweon, I. S., & Sheikh, Y. (2017). *Deltille Grids for Geometric Camera Calibration.* Paper presented at the 2017 IEEE International Conference on Computer Vision (ICCV).

Harris, C., & Stephens, M. (1988). *A combined corner and edge detector.* Paper presented at the Alvey vision conference.

Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision*: Cambridge university press.
Harvey, P. (2013). ExifTool by Phil Harvey.
Heikkila, J., & Silven, O. (1997). *A four-step camera calibration procedure with implicit image correction.* Paper presented at the Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on.
Hellerstein, J. M. (2008). Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*.
Ivosevic, B., Han, Y.-G., & Kwon, O. (2017). Calculating coniferous tree coverage using unmanned aerial vehicle photogrammetry. [journal article]. *Journal of Ecology and Environment, 41*(1), 10. doi: 10.1186/s41610-017-0029-0
Jadejaroen, J., Hamada, Y., Kawamoto, Y., & Malaivijitnond, S. (2015). Use of photogrammetry as a means to assess hybrids of rhesus (*Macaca mulatta*) and long-tailed (*M. fascicularis*) macaques. *Primates, 56*(1), 77-88. doi: 10.1007/s10329-014-0450-2
Jaquet, N. (2006). A simple photogrammetric technique to measure sperm whales at sea. *Marine Mammal Science, 22*(4), 862-879.
Jedlička, J., & Potůčková, M. (2007). Correction of radial distortion in digital images. *Proceedings Technical Computing Prague*.
JEITA. (2002). Exchangeable image file format for digital still cameras: Exif Version 2.2 https://www.exif.org/Exif2-2.PDF.
Jensen, R. J. (2003). The conundrum of morphometrics. *Taxon, 52*(4), 663-671.
. JFoenix. (2015). http://www.jfoenix.com/.
Kannala, J., & Brandt, S. S. (2006). A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(8), 1335-1340.
Kim, J.-S., Gurdjos, P., & Kweon, I.-S. (2005). Geometric and algebraic constraints of projected concentric circles and their applications to camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(4), 637-642.
Kim, J.-S., Kim, H.-W., & Kweon, I. S. (2002). A camera calibration method using concentric circles for vision applications. *ACCV2002, Melbourne, Australia*.
Kim, W., Choi, B.-J., Hong, E.-K., Kim, S.-K., & Lee, D. (2003). A taxonomy of dirty data. *Data mining and knowledge discovery, 7*(1), 81-99.
Kocovsky, P. M., Adams, J. V., & Bronte, C. R. (2009). The effect of sample size on the stability of principal components analysis of truss-based fish morphometrics. *Transactions of the American Fisheries Society, 138*(3), 487-496.
Kraus, K. (2011). *Photogrammetry: geometry from images and laser scans. 2nd edition* (I. A. Harley & S. Kyle, Trans.): de Gruyter.
Letessier, T. B., Juhel, J.-B., Vigliola, L., & Meeuwig, J. J. (2015). Low-cost small action cameras in stereo generates accurate underwater measurements of fish. *Journal of Experimental Marine Biology and Ecology, 466*, 120-126. doi: http://dx.doi.org/10.1016/j.jembe.2015.02.013
Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics, 2*(2), 164-168.

Liu, Y., Liu, S., Cao, Y., & Wang, Z. (2016). Automatic chessboard corner detection method. *IET Image Processing, 10*(1), 16-23.

Luhmann, T., Robson, S., Kyle, S. A., & Harley, I. A. (2006). *Close range photogrammetry: Principles, techniques and applications*. Caithness, Scotland: Whittles Publishing.

Mallison, H., & Wings, O. (2014). Photogrammetry in paleontology–a practical guide. *Journal of Paleontological Techniques*.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics, 11*(2), 431-441.

Maxwell, E. (2017). MVC VS. MVP vs. MVVM on Android. Retrieved from

McFall, G. B., Shepard, A. N., Donaldson, C. L., & Hulbert, A. W. (1992). *Development and application of a low-cost paired-laser measuring device*. Paper presented at the Diving for Science 1992, Wilmington, North Carolina.

Meise, K., Mueller, B., Zein, B., & Trillmich, F. (2014). Applicability of single-camera photogrammetry to determine body dimensions of pinnipeds: Galapagos sea lions as an example. *PLoS ONE, 9*(7), e101197. doi: 10.1371/journal.pone.0101197

Melen, T. (1996). Geometrical modelling and calibration of video cameras for underwater navigation.

Milliet, Q., Delémont, O., & Margot, P. (2014). A forensic science perspective on the role of images in crime investigation and reconstruction. *Science & Justice, 54*(6), 470-480.

Morizur, Y., Ogor, A., & Lespagnol, P. (1994). Bar codes in fisheries research: development of the "ichthyometer". *Aquatic Living Resources, 7*(4), 295-300.

Noakes, D. (2002). metadata-extractor. Retrieved from https://drewnoakes.com/code/exif/

Qi, W., Li, F., & Zhenzhong, L. (2010). *Review on camera calibration.* Paper presented at the Control and Decision Conference (CCDC), 2010 Chinese.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing, 1*(3), 244-256.

Reyment, R. A. (1996). An idiosyncratic history of early morphometrics *Advances in morphometrics* (pp. 15-22): Springer.

Reyment, R. A. (2010). Morphometrics: An historical essay *Morphometrics for Nonmorphometricians* (pp. 9-24): Springer.

Rising, J. D., & Somers, K. M. (1989). The measurement of overall body size in birds. *The Auk, 106*(4), 666-674.

Rohlf, F. J. (1990). Morphometrics. *Annual Review of Ecology and Systematics, 21*(1), 299-316.

Rohlf, F. J., & Marcus, L. F. (1993). A revolution in morphometrics. *Trends in Ecology and Evolution, 8*, 129-129.

Rothman, J. M., Chapman, C. A., Twinomugisha, D., Wasserman, M. D., Lambert, J. E., & Goldberg, T. L. (2008). Measuring physical traits of primates remotely: the use of parallel lasers. *American Journal of Primatology, 70*(12), 1191-1195. doi: 10.1002/ajp.20611

Rufli, M., Scaramuzza, D., & Siegwart, R. (2008). *Automatic detection of checkerboards on blurred and distorted images.* Paper presented at the

Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on.

Sadou, M. C., Beltran, R. S., & Reichmuth, C. (2014). A calibration procedure for measuring pinniped vibrissae using photogrammetry. *Aquatic Mammals, 40*(2), 213-218. doi: 10.1578/AM.40.2.2014.213

Schneider, C. A., Rasband, W. S., & Eliceiri, K. W. (2012). NIH Image to ImageJ: 25 years of image analysis. [10.1038/nmeth.2089]. *Nature Methods, 9*(7), 671-675.

Shrader, A. M., Ferreira, S. M., & Van Aarde, R. J. (2006). Digital photogrammetry and laser rangefinder techniques to measure African elephants. *South African Journal of Wildlife Research-24-month delayed open access, 36*(1), 1-7.

Simek, K. (2013). Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix. Retrieved from http://ksimek.github.io/2013/08/13/intrinsic/

Skiljan, I. (2003). IrfanView. Retrieved from https://www.irfanview.com/

Smith, S. M., & Brady, J. M. (1997). SUSAN—a new approach to low level image processing. *International journal of computer vision, 23*(1), 45-78.

Sprules, W. G., Holtby, L. B., & Griggs, G. (1981). A microcomputer-based measuring device for biological research. *Canadian Journal of Zoology, 59*(8), 1611-1614.

Strobl, K., Sepp, W., Fuchs, S., Paredes, C., & Arbter, K. (2010). Dlr calde and dlr callab. *Institute of Robotics and Mechatronics, German Aerospace Center (DLR)*.

Sugimori, Y., Kusunoki, K., Cho, F., & Uchikawa, S. (1977). Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *The International Journal of Production Research, 15*(6), 553-564.

Suzuki, S. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing, 30*(1), 32-46.

Vo, M. N., Wang, Z., Luu, L., & Ma, J. (2011). Advanced geometric camera calibration for machine vision. *Optical Engineering, 50*(11), 110503.

Weldon, W. F. R. (1890). II. The variations occurring in certain decapod crustacea.—I. Crangon vulgaris. *Proceedings of the Royal Society of London, 47*(286-291), 445-453.

Weng, J., Cohen, P., & Herniou, M. (1992). Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*(10), 965-980.

Whitehead, H., & Gordon, J. C. D. (1986). Methods of obtaining data for assessing and modeling sperm whale populations which do not depend on catches. *Reports of the International Whaling Commission*(Special Issue 8), 149-165.

Whitehead, H., & Payne, R. (1981). New techniques for assessing populations of right whales without killing them. In J. Gordon Clark (Ed.), *Mammals in the Sea. Volume 3: general papers and large cetaceans* (Vol. 3, pp. 189-209). Rome: FAO.

WolfWings, N. v. B. (2010). In P. d. s. a. M. d. s. Barrel_distortion.svg (Ed.).

Xiao, Y., & Fisher, R. (2010). *Accurate feature extraction and control point correction for camera calibration with a mono-plane target.* Paper presented at the Proc. 2010 Int. Conf. on 3D Processing, Visualization and Transmission, Paris.

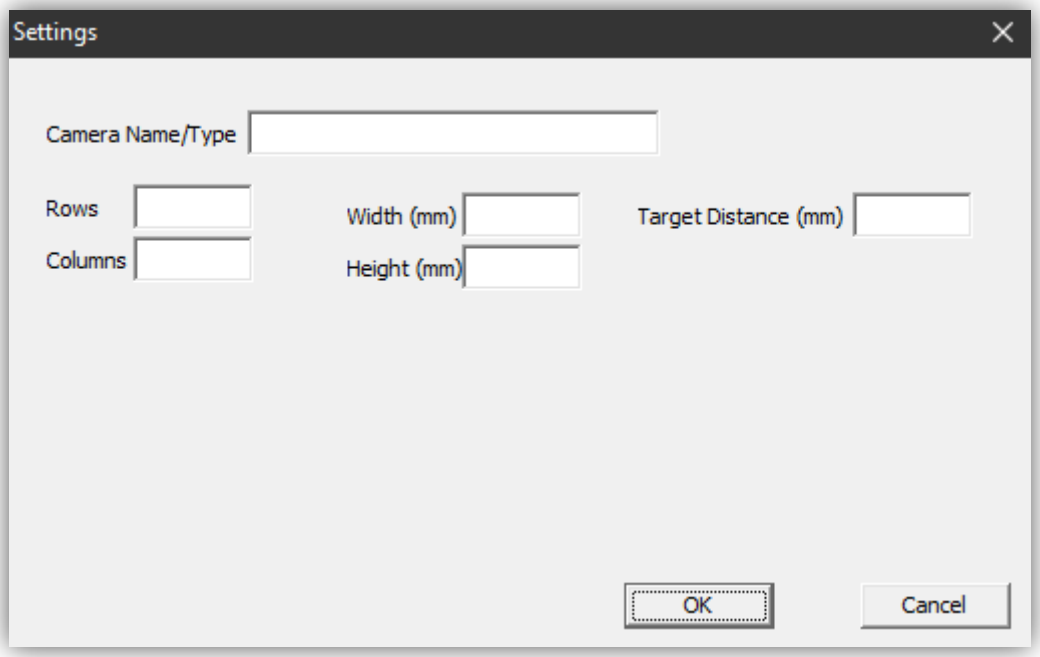
Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(11), 1330-1334. doi: 10.1109/34.888718

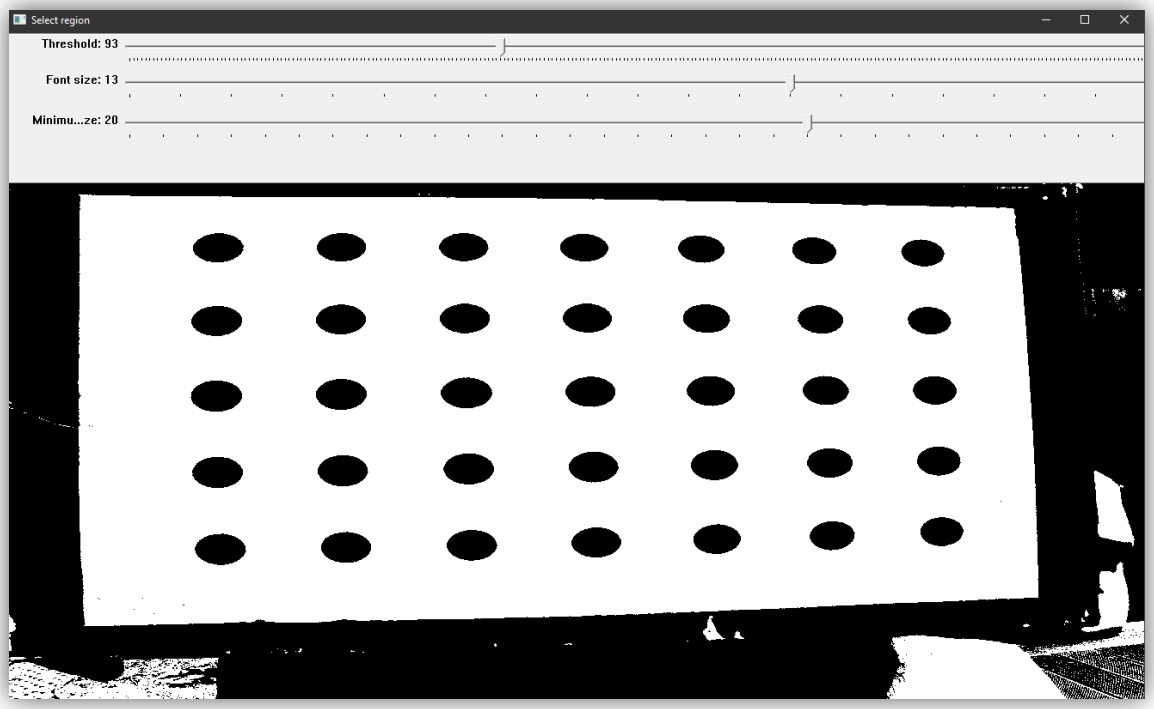
Zou, X., Mõttus, M., Tammeorg, P., Torres, C. L., Takala, T., Pisek, J., . . . Pellikka, P. (2014). Photographic measurement of leaf angles in field crops. *Agricultural and*

Forest Meteorology, 184, 137-146. doi: https://doi.org/10.1016/j.agrformet.2013.09.010

Annexes and Appendices

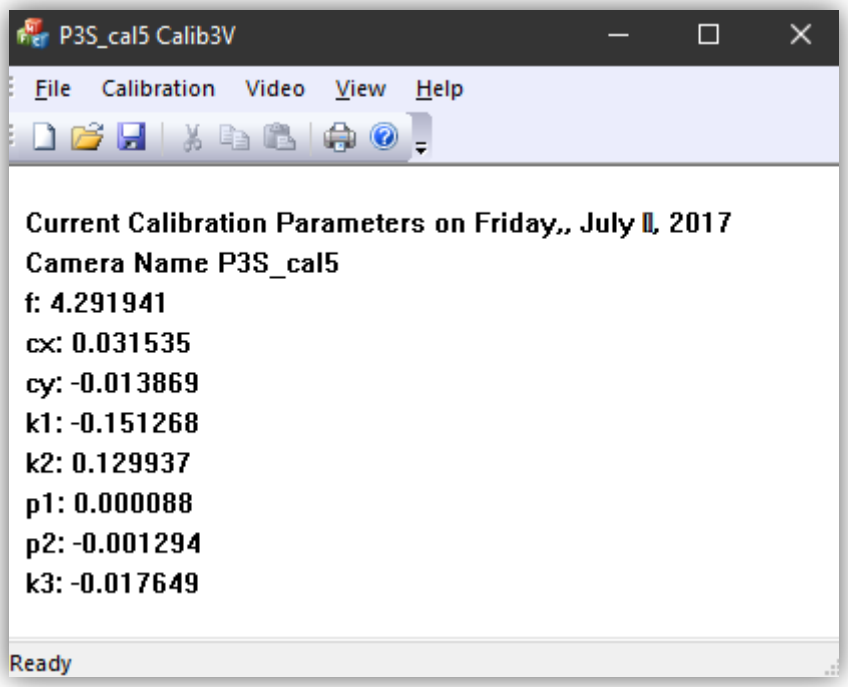
Annex A





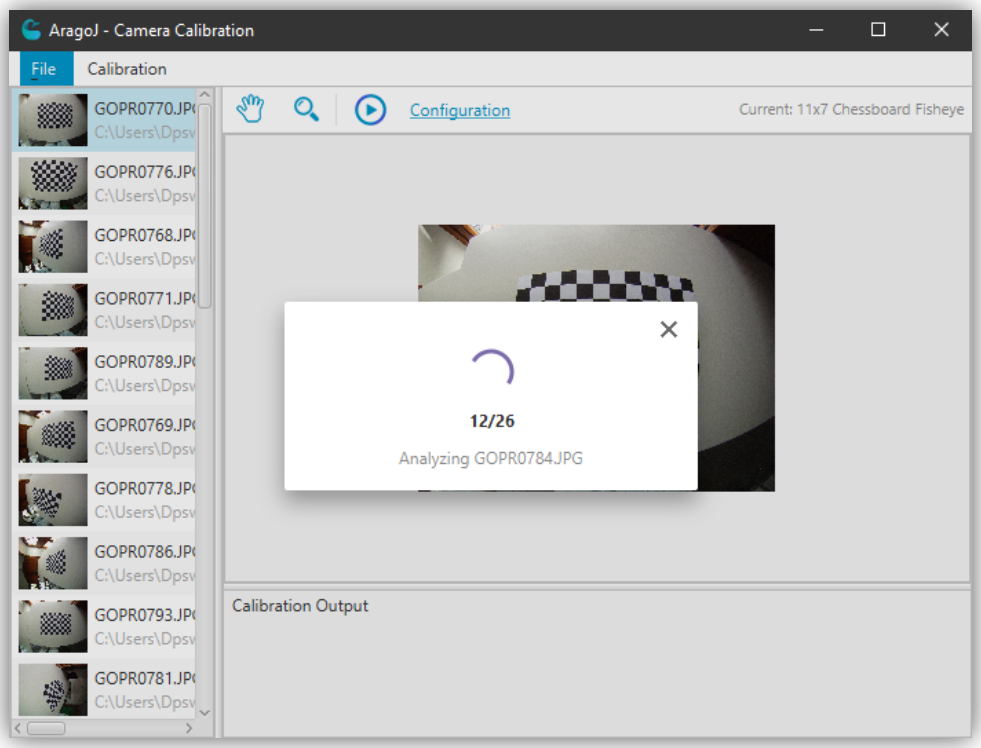
Calib3V (Balletti et al., 2014) calibration settings dialog (top image) region selection for each image (bottom image)

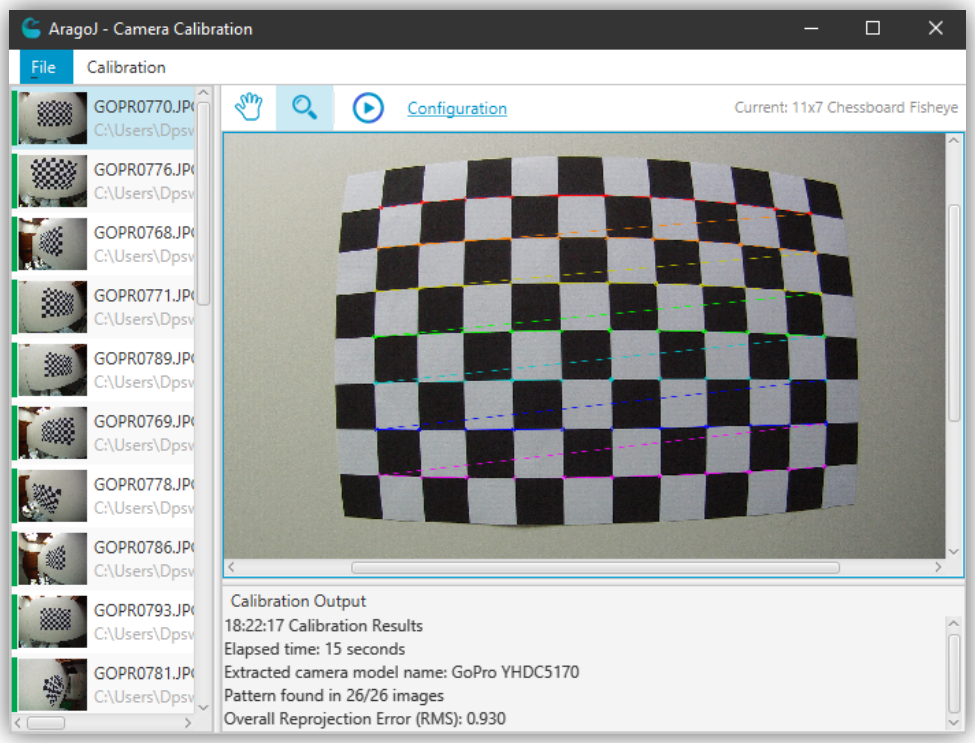
Annex B



Calib3V (Balletti et al., 2014) results of the calibration process, i.e., calibration model

Appendix A





Calibration progress (top image) and results of calibration with calibration output, image status and drawn pattern (bottom image)

Appendix B

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<calibration>
    <name>GoPro YHDC5170_Fisheye</name>
    <lens>Fisheye</lens>
    <intrinsic>1139.2131109819431 0.0 1269.5666663473485
        0.0 1135.8829736939094 964.6004268967597 0.0 0.0 1.0
    </intrinsic>
    <distortion_coefficients>-0.0036215178307375576 0.05863121472958669
        -0.14356484718971552 0.10234313866032874
    </distortion_coefficients>
</calibration>
```

Example of a calibration file structure (.acalib)

Appendix C

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<expressions>
    <item>
        <name>Fluke allometric equation</name>
        <expression>10^(0.9268*log10(pw)+0.605)</expression>
    </item>
</expressions>
```

Example of expressions file structure containing one mathematical expression (.xml)

Appendix D

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<session>
    <image>
        <layers>
            <line>
                <name>Eixo Primário</name>
                <startPointX>1872.94195187489</startPointX>
                <startPointY>916.5321382661155</startPointY>
                <endPointX>2104.351908121347</endPointX>
                <endPointY>1728.494539734386</endPointY>
                <color>0xffeb3bff</color>
            </line>
            <line>
                <name>Eixo Secundário</name>
                <startPointX>1868.0159290650108</startPointX>
                <startPointY>1143.1601330694687</startPointY>
                <endPointX>1984.3997013549779</endPointX>
                <endPointY>1109.9906616694175</endPointY>
                <color>0xf57f17ff</color>
            </line>
            <expression>
                <name>Fluke allometric equation</name>
                <expression>10^(0.9268*log10(pw)+0.605)</expression>
                <variables>
                    <name>pw</name>
                    <value>844.29</value>
                </variables>
            </expression>
        </layers>
        <scaleRatio>
            <ratio>0.9370915841584159</ratio>
            <units>cm</units>
        </scaleRatio>
        <sourceImagePath>C:\...\DJI_0002.JPG</sourceImagePath>
        <zoom>
            <scale>1.3166069453149472</scale>
            <hValue>0.48279630411946967</hValue>
            <vValue>0.33248574387400903</vValue>
        </zoom>
    </image>
</session>
```

Example of a program-specific session file structure (.axml)

Appendix E

```
Source,Modified date,Size,Make,Model,Latitude,Longitude,Altitude
Ref,Altitude,RelativeAltitude,Eixo Primário (px),Eixo Primário
(cm),Eixo Secundário (px),Eixo Secundário (cm),Fluke allometric
equation
C:\Users\...\Desktop\Tese\DJI_0002.JPG,2018/02/15 23:45:27,4.41
MB,DJI,FC300C,38.64998166666667,-28.411974999999998,Below sea
level,132 metres,+25.20,844.29,791.18,121.02,113.41,2076.215633913852
```

Example of a user-specific session file structure with one image (.CSV)