NTNU

Norwegian University of
Science and Technology

Automatic georeferencing of Orthophotographs, and Aerial Images

Sindre Nistad

Master of Science in Engineering and ICT
Submission date: June 2016
Supervisor: Terje Skogseth, BAT
Co-supervisor: Alexander Nossum, NorKart
Trond Arve Haakonsen, Vegdirektoratet

Norwegian University of Science and Technology
Department of Civil and Transport Engineering

Soli Deo Gloria

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF CIVIL AND TRANSPORT ENGINEERING

Report Title: Automatic georeferencing of Orthophotographs and Aerial Images	Date: 10.06.2016 Number of pages (incl. appendices): 232			
	Master Thesis	X	Project Work	

Name:
Sindre Nistad

Professor in charge/supervisor:
Terje Skogseth

Other external professional contacts/supervisors:

Trond Arve Haakonsen, Statens Vegvesen
Alexander Salveson Nossum, NorKart AS

Abstract:
With the advent of drones, the need for rapid, and accurate georeferencing of orthophotos is expected to grow. Per now (June 2016) there are no commercial solutions that is capable of automatically georeference an orthophoto to a set of ground control points. Some semi-automatic solutions exist, but these rely on user interaction, or on having an already georeferenced orthophoto, which will then be matched to the desired orthophoto.

A prototype for such a fully automatic system has been developed as part of this Master's thesis to investigate the possibility, and feasibility of automatic georeferencing of orthophotos, aerial images, with or without a digital elevation model.
By applying the prototype to a orthophoto with a corresponding surface model over Lerkendal, Trondheim, Norway, the program was able to calculate the absolute orientation, and georeference the orthophoto in less than 7 minutes. The root mean square error for the location (Northing-Easting) was as small as 0.0738 meters. Unfortunately, the elevation was not as good (0.847 meters). If the main purpose of georeferencing the data is to only have a referenced orthophoto, this prototype comparable with manual referencing in terms of accuracy, and can be even faster than the manual approach.
The prototype, and it source code is freely available at https://github.com/cLupus/AutoRef/. The source code is licensed under the Mozilla Public License 2.0, which mean you are free to use, modify and distribute all, or part of the source code.

During development, the concept, and algorithm of topological point patterns from Li and Briggs (2006) was adapted to investigate whether it could be used in a different context. It could. However, some faults were found in the presented algorithm. The faults were fixed, and an implementation is given.

Part of this thesis investigate different algorithms for finding the optimal absolute orientation parameters, in a least-squares error sense. Between the method presented in Horn (1987), and Horn et.al. (1988) the difference in root mean square error, and parameters for the absolute orientation was rather small. Umeyama (1991) presents a different algorithm, which is also claimed to be optimal. In particular, the scaling factor presented in the article is claimed to give the minimal root mean square error. However, this is proven to be a false statement in this thesis. The proposed scale factor is sub-optimal compared to using the scale factor proposed in Horn (1987), when everything else is the same.

Keywords:

1. Automatic georeferencing
2. Automatic extraction of ground control points
3. Least-Squares estimation of absolute orientation
4. Topological point Pattern

Summary

With the advent of drones, the need for rapid, and accurate georeferencing of orthophotos is expected to grow. Per now (June 2016) there are no commercial solutions that is capable of automatically georeference an orthophoto to a set of ground control points. Some semi-automatic solutions exist, but these rely on user interaction, or on having an already georeferenced orthophoto, which will then be matched to the desired orthophoto.

A prototype for such a fully automatic system has been developed as part of this Master's thesis to investigate the possibility, and feasibility of automatic georeferencing of orthophotos, aerial images, with or without a digital elevation model. By applying the prototype to a orthophoto with a corresponding surface model over Lerkendal, Trondheim, Norway, the program was able to calculate the absolute orientation, and georeference the orthophoto in less than 7 minutes. The root mean square error for the location (Northing-Easting) was as small as 0.0738 meters. Unfortunately, the elevation was not as good (0.847 meters). If the main purpose of georeferencing the data is to only have a referenced orthophoto, this prototype comparable with manual referencing in terms of accuracy, and can be even faster than the manual approach. The prototype, and it source code is freely available at `https://github.com/cLupus/AutoRef/`. The source code is licensed under the Mozilla Public License 2.0, which mean you are free to use, modify and distribute all, or part of the source code.

During development, the concept, and algorithm of topological point patterns from Li and Briggs (2006) was adapted to investigate whether it could be used in a different context. It could. However, some faults were found in the presented algorithm. The faults were fixed, and an implementation is given.

Part of this thesis investigate different algorithms for finding the optimal absolute orientation parameters, in a least-squares error sense. Between the method presented in Horn (1987), and Horn et al. (1988) the difference in root mean square error, and parameters for the absolute orientation was rather small. Umeyama (1991) presents a different algorithm, which is also claimed to be optimal. In particular, the scaling factor presented in the article is claimed to give the minimal root mean square error. However, this is proven to be a false statement in this thesis. The proposed scale factor is sub-optimal compared to using the scale factor proposed in Horn (1987), when everything else is the same.

Key words: Automatic georeferencing, automatic extraction of ground control points, Least-Squares estimation of absolute orientation, topological point pattern.

Sammendrag

Nå som droner har gjort sin fremtreden, er behovet for hurtig og nøyaktig stedfesting ventet å øke. Da denne masteroppgaven ble skrevet (juni 2016), var det ingen kommersielt tilgjengelige løsninger som kan stedfeste et ortofoto automatisk basert på innmålte fastmerker. Det finnes noen halvautomatiske løsninger, men disse er avhengige av menneskelig innblanding, eller å ha et ortofoto som allerede er stedfestet. De to ortofotene blir så matchet mot hverandre.

En prototype av et slikt fullautomatisk system har blitt utviklet som del av denne masteroppgaven. Målet var å utforske om det i det hele tatt er mulig å stedfeste et ortofoto automatisk ved å finne fastmerkene i bildet automatisk og så stedfeste ortofotoet basert på de innmålte fastmerkene. Dersom et slik system er mulig, var målet å undersøke hvor praktisk og anvendbart et slikt system kan være. Systemet kan brukes på både ortofoto og flyfoto med og uten tilhørende terrengmodell. Et ortofoto, med tilhørende terrengmodell over Lerkendal, Trondheim samt en liste med innmålte fastmerker var gitt som input til prototype. Den var da i stand til å regne ut den ytre orienteringen av ortofotoet med en nøyaktighet på $0,0738$ meter for planet i løpet av 7 minutter. Nøyaktigheten for terrenghøyden var ikke like bra ($0,847$ meter). Om målet er å kun ha et stedfestet ortofoto, så kan denne prototypen sammenlignes med resultatene en ville fått ved å stedfeste ortofotoet manuelt. Tidsmessig, er det muligens hurtigere også. Prototypen og kildekoden til den er fritt tilgjengelig fra `https://github.com/cLupus/AutoRef/`. Kildekoden er lisensiert under «Mozilla Public License 2.0». Det betyr at du er fri til å bruke, endre og distribuere hele kodebasen, eller deler av den.

Under utviklingen av prototypen, ble konseptet og algoritmen om topologisk punktmønster fra Li and Briggs (2006) tilpasset for å undersøke om topologiske punktmønstre kan bli brukt i flere sammenhenger enn det som ble presentert i artikkelen. Det kunne det. Under implementasjonen av algoritmen, ble det oppdaget noen feil og mangler i selve algoritmen. Disse ble rettet og er beskrevet.

Å sammenligne forskjellige algoritmer for å finne den (optimale) ytre orienteringen ved den miste kvadraters metode er en annen del av masteroppgaven. Metodene i Horn (1987) og Horn et al. (1988) gav stort sett lignede resultater, men ikke identiske. Forskjellen mellom parameterne for den ytre orienteringen var også liten. Metoden som er presenter i Umeyama (1991), derimot, gav vesentlig mindre nøyaktige resultater. I artikkelen er det påstått at skaleringsfaktoren som presenteres er optimal, eller minimerer den totale feilen (RMSE). I masteroppgaven vises det at dette ikke stemmer; skaleringsfaktoren gitt i Horn (1987) gir mye bedre resultater (en reduksjon i RMSE på så mye som $99,917\%$), når alt annet holdes likt.

Nøkkelord: Automatisk stedfesting, automatisk ekstrahering av fastmerker fra ortofoto, minste kvadraters metode for ytre orientering, topologiske punktmønstre.

NTNU

Norwegian University of Science and Technology
Faculty of Engineering Science and Technology
Department of Civil and Transport Engineering

Date
15.01.2016

MASTER DEGREE THESIS

Course TBA4925 Geomatics, master thesis
Spring 2016
for
Student: **Sindre Nistad**

Automatic georeferencing of Orthophotographs and Aerial Images

BACKGROUND

An important part of the geomatics industry, and geographical information science is to have data that is georeferenced. There exists some semi-automatic solutions, but these require data that is already georeferenced. With the advent of drones, and airplanes before them, the need to georeference orthophotos, and aerial images in general, is expected to increase. For high accuracy results, ground control points (GCP) are necessary. To find the GCPs in the orthophoto automatically, and then georeference it is therefore of interest.

TASK

The main focus for the thesis is as follows:

To investigate the possibility, and feasibility of automatically finding marked ground control points in an arbitrary image, or orthophoto, and then calculating the absolute orientation of the given image, and the corresponding digital surface/elevation model.

Task description

The task will be accomplished by developing a working prototype for a system that is capable of detecting marked ground control points in an orthophoto, and then applying the developed prototype to match these locations with the measured-in coordinates of the ground control points, and thus calculate the absolute orientation of the orthophoto.

The prototype will be applied to two different orthophotos. For both of them, a digital surface/elevation model, and the measured-in coordinates of the ground control points are available.

Objective and purpose

Create a prototype that is capable of automatically finding ground control points that is signaled.

The prototype, and all accompanying source code is to be open source, and freely available at https://github.com/cLupus/AutoRef. In addition to the prototype, another research questions are if there is a significant difference in the results obtained from different approaches to calculate the absolute orientation, and to investigate if the algorithm presented in Li & Briggs (2006) can be adapted to solve the problem of matching the set of candidates extracted from the orthophoto with the measured-in coordinates of the ground control points.

Startup and submission deadlines

Startup: January 15th 2016. Submission date: Digitally in DAIM at the latest June 10th 2016.

Supervisors

Supervisor at NTNU: Terje Skogseth
Co-supervisors: Trond Arve Haakonsen, Statens vegvesen
Co-supervisors: Alexander Salveson Nossum, Norkart AS

Department of Civil and Transport Engineering, NTNU. Date 15.01.2016 (revised June 2016).

Terje Skogseth (signature)

Preface

MANY THANKS to Terje Skogseth, Trond Arne Haakonsen, and Alexander Salveson Nossum for supervision, guidence, and many advice.

Thanks are also due to the Norwegian Public Roads Administration and Norwegian University of Science and Technology for letting me use their images over E6, and Lerkendal, Trondheim (respectively), along with Trond Arve Haakonse, and Terje Skogseth for establishing, measuring in, and correcting the ground control points used at Lerkendal and E6.

Many thanks to my parents, Eilif Nistad, and Marit Elisabeth Nistad for their support and encouragements through my five years at Norwegian University of Science and Technology, and for all the time before that.

Thanks also to friends of old and new.

Notes

BEFORE UNDERTAKING this thesis, there are some notes, or conventions that the reader should be aware of.

Note on abbreviations and acronyms

The full name of an abbreviation is repeated at least once per chapter.

If the last word of the abbreviation is a noun, it will not be repeated when that noun is referred to, or used. Rather, only the abbreviation is used when the acronym is well known, while the full phrase is used when it is not.

An example would be the use of "Global Navigation Satellite System (GNSS)". In this thesis, a particular Global Navigation Satellite System, such as GALILEO, will be referred to as "The GNSS GALILEO is an European satellite system for navigation", and *not* "The GNSS-system GALILEO ...". Had GNSS not been well known (in the Surveying profession), the same sentence could be read as "The Global Navigation Satellite System GALILEO is an European ...". Note that the abbreviation is not included in parenthesis. This will be the case, even if it is the first usage. The abbreviation will be given in the second instance instead.

Note on Figures

All figures have been exported from the program they were created in to the Portable Network Graphics format. Any excess margins were removed by using the Trim tool in Adobe PhotoShop CC (2015). This tool is capable of detecting, and removing the excess margins automatically. The reason for trimming them automatically, is that images exported from MATLAB, in particular, had a tendency to include excessive margins.

Any Figure (e.g. photograph or illustration) that is not accredited, is made by the author for the thesis. Some are made by using programs such as InkScape, Adobe PhotoShop, MATLAB, and Edraw Max.

The Portable Network Graphics format was chosen because it uses lossless compression, works well with LaTeX, MATLAB, and Portable Document Format-files.

Images that took up more than approximately 100 MB, such as orthophotos were stored in Tagged Image File Format (TIFF). The reason for this, is flexibility. Images can be compressed with, or without loss, or with no compression at all. With large images, no compression can be advantageous, as the image can be loaded into a program faster, then if the image had to be decompressed first.

Tags, such as location, and absolute orientation can be stored directly in the image, which allows more convenient storage of the georeference. This can also be done for Joint Photographics Experts Groups (JPEG) images, but JPEG cannot be stored without the use of lossy compression.

The standard TIFF does not support file sizes that exceeds 2^{32} bytes, or 4 GB. This can be amended by using the now standardized BigTIFF, which has a maximum limit of 2^{64} bytes, or 16 777 216 TB. In comparison, the largest hard drive to date (May 2016), is a 16 TB Solid State Disk from Samsung (Zhang, 2015). The largest image that was encountered during the work on this thesis, was about 21 GB. In practice this limit will not be any problem for the foreseeable future.

BigTIFF is backwards compatible with TIFF. The reverse need not be true, however. Many applications does support BigTIFF. MATLAB is one example.

Note on words in the thesis

The words "computer program", "program", "application", "*the* implementation", and "system" will be used interchangeably. They refer to the source code in Appendix A. However, the word "implementation", will also be used for implementations of other programs, or applications.

A working name for the program is "AutoRef".

The words "geomatics", "surveying", and "geography" are used interchangeably for the profession, and community of surveyors, GIS-analysts, geographers, and more...

Note on mathematical notation

All *vectors* are assumed to be column vector, and are written as lowercase letters in bold. When assigning a vector, it is enclosed by square brackets. An example is $\mathbf{a} = [1, 2, 3, 4]^\intercal$, where $^\intercal$ is the transpose operator.

The *norm*, or length of a vector is defined as $||\boldsymbol{x}|| = \sqrt{x^\intercal x} = \left(\sum_{i=1}^{n} (x_i \cdot x_i)^2\right)^{\frac{1}{2}}$.

A *point* is denoted as the vector $\boldsymbol{p}_{name,index}$, where *name* denotes which set it belongs to, while *index* is the index of the particular point in a particular set. To differentiate them, a point will use parenthesis, while vectors use square brackets in assignments. For example, $\boldsymbol{p}_{x,i} = (4352, 495, 100.43)$. When convenient, points are considered vectors that go from the origin to the points them self.

Matrices will be written as uppercase letters, also in bold. The matrix is enclosed in parentheses. For example $\mathbf{I} = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$. A matrix can also be referred to as $\boldsymbol{A} = (a_{ij})$. The size of the matrix is then given. The element a_{ij} is the i^{th} column on the j^{th}.

Scalars are written as lowercase letters, such as $a = 4\pi$.

Sets are denoted by a non-bold capital letter, such as $A = \{1, 3, 4, 2\}$. They are assumed to be unordered, and all elements are assumed to be unique.

Collections are or sets of sets, and are denoted by a calligraphy, non-bold capital letter. An example is $\mathcal{P} = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$.

A *sequence* is similar to a set, except that it has an order. Both have only unique elements. Sequences are denoted in capital Gothic letters, and angled brackets. An example is $\mathfrak{T} =$

$\langle 1, 2, 3, 4 \rangle$.

Sets that are arbitrarily large, such as the real numbers, and the integers, are denoted by double letters. Examples include $\mathbb{Z}$, and $\mathbb{R}$, which is the set of all integers, and all real numbers respectively. $\mathbb{Z}^+$ will denote all positive integers, not including 0, while $\mathbb{N}$ is the set of all natural numbers, which is all positive integers in addition to 0.

The symbol $\sim$ is used to denote correspondence. That is, if $\boldsymbol{p}_{x,i} \sim \boldsymbol{p}_{y,i}$, then the point $\boldsymbol{p}_{x,i}$, which might be a model coordinate, is equivalent to the point $\boldsymbol{p}_{y,i}$, which might be a measured-in ground control point (GCP).

Angles are assumed to be in *radians*, unless otherwise noted.

For absolute orientation, $\boldsymbol{t} = (x_a y_a z_a)^\intercal$ is the transnational offset of the image, or model to be orientated. s is the scale factor for the image, or model. $\boldsymbol{R}$ is the rotational matrix, in which the three rotational parameters ϕ, θ, and κ (roll, pitch, and yaw respectively).

In this thesis roll is assumed to be rotation about the principal axis of a model, or image. That is, rotation about the negative y-axis of an image. The negative y-axis is chosen because it is a very common indexing scheme for image (Gonzalez and Woods, 2008a).

Pitch is assumed to be rotation about the secondary axis of the model, or image. That is, rotation about the x-axis of an image.

Yaw is rotation about the axis normal to the image, and in the same direction as the digital elevation model (DEM).

In Kraus (2007), x_a is called X_u, y_a is Y_u, z_a is Z_u, s is m, κ is K, θ is Ω, and ϕ is Φ.

Contents

List of Tables

List of Figures

List of Source Code

Glossary

Absolute Orientation is an affine transformation transformation of an image. More specifically, it is a Similarity Transform that consists of seven parameters (in three dimensional (3D)); three for translation (x, y, z), three for rotation (ϕ, θ, κ, known as roll, pitch, and yaw respectively), and one for scaling, s. Normally the three rotations are not found explicitly, but rather they are implicit in a rotation matrix $\boldsymbol{R}$. The rotations are relative to the model. For a two dimensional orientation, only 4 parameters are required; one rotation, κ, two translations, x and y, and one for scaling; s. vi, x, xi, xvii, xviii, xxii, xxiii, xxv, xxvi, xxvii, xxviii, 1, 2, 4, 12, 18, 19, 23, 26, 29, 30, 31, 32, 35, 43, 45, 48, 59, 60, 61, 62, 67, 68, 69, 86, 98, 99, 100, 112, 138, 148, 163, 164, 165, 166, 167

Absolute Orientation Parameters is a set of parameters that defines the absolute orientation of an image. In 3D, there are seven parameters; three for translation, three for rotation, and one for scaling. In two dimensional (2D), there are four; two for translation, one for rotation, and one for scaling. The parameters are denoted as $\boldsymbol{t}$, $\boldsymbol{R}$, and s respectively. xi, xvii, xviii, xxvii, xxviii, 2, 4, 18, 19, 23, 26, 29, 32, 35, 43, 45, 48, 60, 61, 62, 68, 69, 163, 164, 165, 166, 167

Aerial Image is an image taken of the ground from an aerial vehicle. xxvii, 1, 2, 13, 29, 47, 64

Aerial Vehicle is motorized vehicle that flies, or is otherwise airborne. In this thesis it will refer to drones, manned helicopters, and planes, whose primary purpose is to capture images of the ground. xxv, xxvi, xxvii, 3, 9

Affine Transformation is any transformation that conserves linearity, and ratios of distances. In other words, if three points lie on a line, they will still form a line after the transformation. If one line is twice as long as another, the first will still be twice as long after the transformation. Examples of affine transformations include rotation, reflection, translation, shearing, and scaling (Weisstein, 2016). xxv, xxviii

Bundle Adjustment is an algorithm for simultaneous finding the optimal 3D location of a

structure as seen in multiple images, and the position, orientation, and calibration of the different cameras that took the images. This algorithm, and derivatives thereof, are used to reconstruct a 3D model of what is captured in a set of ground control points (GCPs). From the model, a true orthophoto can be made (Triggs et al., 2000). There are many variations of this method. The main difference between them lies in the choice of error function, and error model (Triggs et al., 2000). x, 7, 9, 29

Candidate Matching is a $n \times 2m$ matrix whose first m columns are image coordinates, while the latter m are Real-World (measured-in) coordinates. One row represents a single point. Each row, then, is a mapping from image coordinates to measured-in coordinates of GCPs, and vice versa. xii, xiii, xiv, xxi, xxviii, xxxi, 2, 18, 29, 32, 45, 60, 102, 148, 149, 150, 151, 152, 153, 154

Creative Commons Attribution-ShareAlike 3.0 is a free culture license that gives anyone the right to use and adapt the work, as long as an attribution to the original creator is given, and that the work is distributed under a similar license. The full license is available at `https://creativecommons.org/licenses/by-sa/3.0/legalcode`. 75

Digital Terrain Model is an elevation model that measures the elevation from a specified geoide to the bare terrain. That is, trees, houses, and any other structure that is not the ground is not included in this model. xxvi, xxxi, 2

Digital Surface Model is a superset of digital terrain model (DTM). It also includes everything that is above ground, such as trees, buildings, and bridges. xxi, xxvi, xxxi, 1

Digital Elevation Model is an umbrella-term for both DTM, and digital surface model (DSM). viii, xxi, xxxi, 18, 30, 35, 43, 69, 104

Drone is an umbrella term for any unmanned aerial vehicle. The main usage of "drones" in this thesis, is in the context of relatively small, battery powered micro-planes, quad-copters, helicopters, and other multi-copters. xxv, 1, 5, 9, 36, 47

Exchangeable Image File Format is a standard for storing meta-data inside an image. Information that can be stored include location, date, and time, who took it, what compression algorithm is used, and more. Only Joint Photographics Experts Groups (JPEG) and Tagged Image File Format (TIFF) support this format. xxxi, 32

GALILEO is an European Global Navigation Satellite System (GNSS). It is a civilian system, unlike GPS and GLONASS which are owned, and operated by military agencies. GALILEO aims to be a completely independent system, while still offer full interoperability with GPS and GLONASS. As of this writing (May 2016), GALILEO is not yet fully operational (European Global Navigation Satelite Systems Agency, 2016). vi, xxvii

Georeference is the process of finding the parameters that gives the absolute orientation of an image. A georeferenced image, or orthophoto is an image for which the absolute orientation is known, and has been applied to the image. In other words, every pixel of the image has a specific Real-World position associated with it. i, vi, xi, 1, 2, 3, 5, 7, 8, 9, 17, 33, 35, 43, 45, 47, 49, 51, 53, 55, 57, 59, 67, 69

Global Positioning System is a constellation of 31 satellites in 6 different medium Earth orbits, with an altitude of approximately 20 200 km. This gives global, real-time 3D positioning and navigation, velocity and timing. It is developed, and owned by the United States Department of Defense. It is operated, and maintained by the United States Air Force. Global Positioning System (GPS) is one of multiple implementations of a GNSS (Nahavandchi, 2015; GPS.gov, 2016). xxvii, xxxi, 9

Global Navigation Satellite System is an umbrella term for any system of satellite constellations that gives the user the possibility of finding his, or her 3D position, velocity, and timing that can be used for navigation and positioning, amongst other applications. The most well-known examples of a GNSS is the United State's GPS, the Russian GLONASS, the European GALILEO, and the Chinese BeiDou. The Indian Regional Nagivation Satellite System is considered to ba a GNSS, even though it only covers India (GNSS Asia, 2015). vi, xxvi, xxvii, xxxi, 9

GLONASS is a GNSS developed, owned, and maintained by the Russian Federation. It consists of 24 satellites in a constellation. Its full name in English is GLObal NAvigation Satellite System, not to be confused with Global Navigation Satellite System. The abbreviation GLONASS will therefore be used. If the full name is refered to, the Russian name will be used in stead to distinguish it from GNSS. The Russian name is "Globalnaya navigatsionnaya sputnikovaya sistema" xxvi, xxvii

Horn refers to the least-squares estimation (LSE) solution for finding the absolute orientation parameters proposed in Horn (1987). xi, xviii, xix, 35, 43, 45, 48, 59, 60, 61, 167

Horn-Hilden refers to the LSE solution for finding the absolute orientation parameters proposed in Horn et al. (1988). xi, xviii, xix, 35, 43, 45, 48, 59, 60, 61, 167

Image will, in this thesis, be a collective term for orthophotos and aerial images. Depending on the context, binary images, and gray-scale images can be any image, and not just those from an aerial vehicle, or images that are orthophotos. vi, viii, x, xiii, xiv, xv, xvii, xxi, xxiii, xxv, xxvi, 1, 2, 3, 5, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 24, 26, 29, 30, 31, 32, 34, 36, 38, 39, 40, 45, 47, 48, 61, 62, 66, 67, 143, 145, 149, 150, 151, 152, 153, 154, 156, 157, 158, 159, 161, 162, 163

L*a*b* is a color model developed by the International Commission on Illumination in three dimensions; Lightness, a, and b. a is goes from yellow to blue, while b goes from red to green. This is because yellow and blue are complementary colors, as is red and green(Cruse, 2015). x, xviii, 12, 39, 44, 48

Least-squares estimation is a method for finding the parameters of a function such that the total error of the given sample data is minimal. ix, xxvii, xxxii, 1, 12, 19, 43, 67, 69, 98

MATLAB® is a programming language, and an integrated development environment (IDE) produced by Swedish MathWorks®. vi, vii, 24, 39, 40, 44, 47, 77, 83, 105, 110, 140, 145

Mozilla Public License version 2.0 is an open source license, which encourage other people to share, and contribute to a project. It also encourage to modify the project and include it into other projects, be they open source, or proprietary. For the full license, see Appendix C. xxxii, 27

NTNU-Geomatics is a group at the Department of Civil and Transport Engineering, Faculty of Engineering Science and Technology at the Norwegian University of Science and Technology (NTNU) 36, 38

Orthophoto is an image that was been processed such that it is orthographic, i.e. any artifacts from a central projection is removed. i, vi, x, xi, xiii, xiv, xv, xvii, xviii, xix, xxiii, xxv, xxvi, xxvii, 1, 2, 3, 5, 7, 8, 9, 13, 15, 18, 29, 30, 32, 33, 35, 36, 38, 39, 43, 44, 45, 46, 48, 47, 48, 59, 62, 63, 64, 66, 67, 69, 142, 145, 149, 150, 151, 152, 153, 154, 156, 157, 158, 159, 161, 162, 163, 167, 193

RGB is a device dependent color model in three dimensions; red, green and blue. It is used to display color on most screens, and it is used in most cameras to capture light. Usually 8 bits are used to encode each color, giving approximately 16.8 million unique colors (Rouse, 2005). x, xxxii, 8, 11, 39, 44, 145

Root Mean Square Error is a measure of the total error in an estimation. Defined mathematically as $\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$, where n is the number of observations, and e_i is the difference, or error, between the observed value and the estimated value for data point i. xiv, xxiii, xxxii, 2, 26, 30, 60, 67, 69, 136, 156

Similarity Transform is a linear transformation that allows rotation about an arbitrary axis, translation, and scaling. A Similarity Transform is an affine transformation. The reverse need not be true. xii, xxi, xxv, xxxii, 60, 100, 163

Topological Point Pattern is a concept, and an algorithm for finding a candidate matching (CM) between two sets of points. The concept, and the algorithm will be studied in grater detail in Section 2.5. i, ix, x, xi, xxiii, xxxii, 3, 7, 16, 32, 45, 66, 101

Umeyama refers to the LSE solution for finding the absolute orientation parameters proposed in Umeyama (1991). xi, xviii, xix, 35, 43, 45, 48, 60, 61, 62, 63, 67, 167

Umeyama* refers to a modified version of the method for finding the absolute orientation parameters proposed in Umeyama (1991). The scale factor, s, is calculated same way it is in Horn (1987) and Horn et al. (1988) instead of how it was originally calculated in Umeyama (1991). xviii, xix, 35, 43, 45, 48, 61, 167

World file is an auxiliary file for any image that describes the placement, and orientation of the image in a certain coordinate system. The system can be global, such as UTM, or WGS84, regional, or local. The file name is the same as the image with the extension .*w, where * is the first two consonants of the image's file extension. If an image is a called "ortho.tif", then the world file is called "ortho.tfw". xxiii, 32, 139

Acronyms

2D two dimensional xxv, 19, 25, 29, 43, 64, 67, 69

3D three dimensional xxv, 2, 5, 11, 19, 20, 29, 30, 31, 43, 64, 67, 69

BAT Department of Civil and Transport Engineering, Faculty of Engineering Science and Technology at the Norwegian University of Science and Technology xxviii, 4

CM candidate matching *Glossary:* candidate matching, xii, xiii, xiv, xv, xxi, xxviii, 2, 18, 29, 32, 45, 60, 67, 68, 102, 148, 149, 150, 151, 152, 153, 154, 156, 157, 158, 159, 161, 162, 163, 167

DEM digital elevation model *Glossary:* Digital Elevation Model, viii, xxi, 18, 30, 35, 38, 43, 69, 104

DSM digital surface model *Glossary:* Digital Surface Model, xxi, xxvi, 1

DTM digital terrain model *Glossary:* Digital Terrain Model, xxvi, 2

EXIF Exchangeable Image File Format *Glossary:* Exchangeable Image File Format, 32

GCP ground control point v, viii, ix, x, xi, xii, xiii, xiv, xv, xvii, xviii, xix, xxi, xxii, xxiii, xxv, xxvi, 1, 2, 3, 10, 12, 13, 15, 16, 17, 18, 19, 22, 24, 26, 29, 30, 31, 32, 34, 35, 36, 38, 39, 41, 43, 44, 45, 46, 47, 48, 47, 48, 59, 60, 61, 62, 63, 64, 66, 67, 68, 69, 86, 100, 101, 103, 110, 136, 142, 145, 146, 149, 150, 151, 152, 153, 154, 156, 157, 158, 159, 161, 162, 163, 167, 193

GIS geographic information system 3

GNSS Global Navigation Satellite System *Glossary:* Global Navigation Satellite System, vi, xxvi, xxvii, 9

GPS Global Positioning System *Glossary:* Global Positioning System, xxvi, xxvii, 9

IDE integrated development environment xxviii

IR infrared 8

JPEG Joint Photographics Experts Groups vi, xvii, xxvi, 32, 39

LSE least-squares estimation *Glossary:* Least-squares estimation, ix, xxvii, xxviii, 1, 2, 12, 19, 43, 67, 69, 98, 99

MPL 2.0 Mozilla Public License Version 2.0 *Glossary:* Mozilla Public License, 27

NPRA Norwegian Public Roads Administration v, 1, 36

NTNU Norwegian University of Science and Technology v, xxviii, xxxi, 1, 4, 35

PNG Portable Network Graphics vi

RGB red green blue *Glossary:* RGB, x, xvii, xviii, 8, 11, 39, 44, 48, 145

RMSE root mean square error *Glossary:* Root Mean Square Error, xiv, xv, xvi, xxiii, 2, 26, 30, 60, 61, 63, 67, 69, 136, 156, 157, 158, 159, 160, 161, 162, 163

RTK real time kinematics 9

ST similarity transform *Glossary:* Similarity Transform, xii, xxi, xxiii, 60, 100, 136, 139, 163

SVD sigular value decomposition 25, 26, 136

TIFF Tagged Image File Format vi, vii, xxvi, 32, 40

TPP topological point pattern *Glossary:* Topological Point Pattern, i, ix, x, xi, xxiii, 3, 7, 9, 12, 16, 17, 18, 32, 45, 66, 67, 101, 116, 128

Chapter 1

Introduction

UP UNTIL now, finding, and matching ground control points (GCPs) in an orthophoto has been a manual process. The same is true for aerial images in general. Much of this thesis is dedicated to the design and development of such a computer program.

Previously, marked measured-in GCPs had to be found manually in the orthophoto, often with the help of a map. The user would then assign the a GCP to one in the orthophoto. The computer on which this was done, would then calculate the absolute orientation of the orthophoto, and georeference it. ArcGIS is such a program.

Now, by giving an orthophoto, or an aerial image and a set of measured-in GCPs to a program, the absolute orientation of the image is found completely automatically; the user need only start the program, and select the input data. The application does the rest.

"Words are wind" is an expression. Therefore, a part of this thesis is dedicated to testing, and verification. That is also why all source code, and data used is disclosed.

The testing was done with realistic, Real-World data from Norwegian University of Science and Technology (NTNU) and the Norwegian Public Roads Administration (NPRA). The data consists of a multitude of aerial images taken from an FylSense eBee drone. The images where then stitched together to form two different orthophotos. They where then given as input along with the measured-in GCPs, and a digital surface model (DSM) when available.

The goal of this program, and subsequent testing, is to investigate the potential and feasibility of a program that can automatically georeference an image by using the image itself and the GCPs. For diversity of usage, one of the datasets was collected for purely scientific purposes, while the other was gathered for government usage.

The experimental setup is discussed in Section 3.3. The development is described in Chapter 3 and 4. Afterward, the results from the experiments are analyzed, and discussed in Chapter 5.

1.1 Goal of the thesis

The overarching goal of this thesis is to investigate the possibility, and feasibility of automatic georeferencing. More specifically, this will be done by automatically detecting marked ground control points (GCPs) in a given orthophoto, or aerial image, extracting these, and matching them to a set of given measured-in GCPs. When a matching is found, the absolute orientation is found by using a least-squares estimation (LSE) algorithm on these two sets of points.

As this is an investigative approach, a prototype is developed. It need not be ready for commercial production, or be blazingly fast, only prove it is possible, and feasible, i.e. takes a reasonable amount of time to compute. Being a prototype, we limit the amount of possible ways the GCPs are marked. The marker this thesis will focus on is a square colored in a safety orange color. An example can be seen in Figure 3.6 and 3.7.

In Section 2.1, we see that some existing solutions claim to be automatic, but require some user input during the process. This is not the case for the program of this thesis. The only user interaction required, is to start the program, and select the input. The solution would then, in a reasonable amount of time, find the absolute orientation of the orthophoto, and give the user the absolute orientation parameters directly, write it to a file, or write the parameters directly into the image file. This way other programs, or software packages can analyze the image further, as spatial information. If the user have a digital terrain model (DTM), and the GCPs are measured-in with Northing, Easting, and Height, and if the user wish to, the program can give a three dimensional (3D) absolute orientation.

In summary, in this thesis we want to develop a prototype that is able to automatically georeference orthophotos and aerial images that have their GCPs marked with an orange square. This prototype is then to be tested on Real-World data.

Supplementary to this, we also want to make the prototype efficient and open source. We want to repeat parts of what was done in Li and Briggs (2006), and to compare three different methods for computing the absolute orientation parameters for the orthophotos.

1.1.1 Secondary Goals

Now that the overarching goals have been formulated, we take a look at some of the minor objectives of this thesis. These will not be the main emphasis, but rather supplement the thesis. Additionally, they will contribute to society at large, and science in particular.

1.1.1.1 Open source

In Section 2.1, we see that similar solutions already exist. These cannot georeference an image to a set of points, but rather to an already georeferenced image, or a to a georeferenced vector set of a road network. One of them have published its pseudo-code for the algorithm, but not the actual source code. The other solutions are proprietary, however. As such, they are of less use to the public domain. Therefore, another goal of this thesis is to make the prototype open source, and publicly available.

1.1.1.2 Multiple LSE techniques

LSE will be discussed in greater detail, and compared against each one another in Section 2.6.1. A selection of implementations of LSE that estimates the absolute orientation parameter are presented in Section 2.6. These methods will be run on the same candidate matchings (CMs), and the result will be compared. In particular, the resulting root mean square error (RMSE) will be compared and analyzed.

1.1.1.3 Investigate the usability of topological point pattern

On of the strongest assets of science, and scientific work is repeatability, and reproducibility. Li and Briggs presents the concept of topological point pattern (TPP), along with an algorithm for matching points. The source code, and data that was used in their article, was not disclosed.

As part of this thesis, aspects of their experiment are repeated; two sets of points are to be matched. One of the sets comes from an image, while the other has its origin in a vector set (points). In this case, all the source code, and data used is disclosed to the scientific community.

1.1.1.4 Effectiveness

If the solution proposed in this thesis, works perfectly, and is incredibly accurate, but takes many days to run, it is, in many ways, a failure. A solution need to be efficient enough in order to be useful. In practice, this might mean that it is faster, or near as fast as the manual alternative.

Even though the solution in this thesis is a prototype, a secondary goal is that it can georeference an orthophoto in a reasonable amount of time.

1.1.1.5 Investigate marks for GCPs

At the time of writing (May 2016), there is no standard for how a GCP should be marked when it is to be captured from an aerial vehicle in Norway. Since automation becomes more and more common, it stands to reason that the future for georeferencing also is automatic. Consequently, it is of interest to investigate what impact the shape, size, and color has for the ease of detection.

Look at what role the markings have, and come with some thoughts, and experiences about it.

1.2 Motivation

In Section 1.4, we see that the Geomatics, and Surveying profession has been eager to use, and adapt new technology, such as drones, and computers when they became available. Yet, as we see in Section 2.1, there are no commercially available solutions, be they proprietary or open source, to the problem of automatically georeference an orthophoto based on its GCPs.

Society at large is leading towards ever more automation. First heavily manual work was automated, then more and more routine tasks were done by machines. Since georeferenc-

ing of orthophoto is a fundamental part of geographic information system (GIS), and a routine task, it is only natural for it to be automated as well.

During the preliminary research for this thesis, and accompanying program, the author has been in contact with representatives from the geomatics, and surveying professional community in Norway, professors, and faculty at Department of Civil and Transport Engineering, Faculty of Engineering Science and Technology at the Norwegian University of Science and Technology (NTNU). All of these was very positive to a system as described here. It was also their opinion that there is a need for such a system as well.

1.3 The structure of the thesis

In the rest of the introduction, we will take a little look at the history of surveying, and see that this work is a natural next step in Surveying, and Geomatics.

The next chapter is concerned with the underlying theory that the program is based upon. The theory is not limited to the program, however. Software packages, and solutions that are similar to the proposed solution, are presented, discussed, and compared against each other.

Chapter 3 presents the design process of the prototype along with how that data that was used was collected and processed. We also go through how the prototype is tested and verified.

In Chapter 4, the results from two Real-World cases are presented. These results are then analyzed and discussed in Chapter 5, before a conclusion about the program's feasibility and usability is drawn in Chapter 6.

The Appendices contain the source code for the program along with URLs to the public repository of the code base, to the data that was used in the development and testing of the program.

1.4 A small history lesson

Surveying has been a part of civilizations since ancient times. One of the first known uses of surveying equipment is in the Middle East, about $5\ 000$ years ago and in China, about $3\ 000$ years ago (Skogseth and Norberg, 1998a). The main use for such equipment, and the corresponding profession, seems to have been much the same as the use is today; to establish, and maintain borders (Land surveyors, 2010). Up until the 14^{th} Century, the equipment used did not change much. Leveling techniques and equipment were developed, and invented in the 16^{th} Century. While Galileo turned his telescope towards the heavens above, surveyors turned theirs toward bench marks and determined heights. (Skogseth and Norberg, 1998b; van Helden, 2016)

In the time between 1920 and 1980, the development of surveying equipment saw a large increase in accuracy, and a drop in both weight and price (Skogseth and Norberg, 1998b). Before this, the development was rather slow, but steady.

In 1980 - 1990, computers were introduced to the world of surveyors. They could easily

do the lengthy computations, such as relative, and absolute orientation, faster, more consistent, and with more accuracy than the surveyor could do. This is also when closed-form solutions for the absolute orientation parameters were first described (Horn, 1987; Arun et al., 1987; Horn et al., 1988).

Later, in our own decade, we have seen software packages, such as PhotoScan, Pix4D, and DroneDeploy, becoming publicly available. These software packages are able to take a number of images and make a 3D model from them. An orthophoto can then be made from the 3D model. With the availability of such software packages, drones have become popular in the field. One of the reasons for this, is that the drone can take pictures over a given area relatively quickly at, almost, any spatial resolution.

If the individual images fed into such software packages have a location associated with it, the programs are able to georeference the orthophoto (AgiSoft LLC, 2012; startupticker.ch, 2014; Kolodny, 2014). There are certain issues associated with this method of georeferencing. These will be discussed in Section 2.1.5.

Considering that a trend in modern history is automation, it then seems natural for an important routine task to be done automatically. For the geomatic sector, one such advancement would be the program developed in this thesis (Ramebäck, 2003).

Chapter 2

Theory

"We are like dwarfs sitting on the shoulders of giants. We see more, and things that are more distant, than they did, not because our sight is superior or because we are taller than they, but because they raise us up, and by their great stature add to ours."

John of Salisbury, *Metalogicon*

ALL OF the concepts, and theoretical frameworks that were used for the implementation, and the subsequent testing and validation of the prototype is well-known to the scientific community at large. Some of it lies outside the normal realm of the geomatics sector, however.

This chapter intends to lay the foundation of what the prototype is built upon.

2.1 Existing solutions

There are software packages that are, to some extent, capable of automatic georeferencing, although they do not solve the problem depicted in the introduction. These include ArcMap, AutoSync, PhotoScan, Pix4D, and DroneDeploy. Another solution was published by Li and Briggs (2006). Unlike the other solutions, Li and Briggs (2006) is not (to the Author's knowledge) commercially available.

The first two work by using an already georeferenced image. The latter uses topological point pattern (TPP). The rest uses a combination of bundle adjustment, and the location associated with its input images.

In the following subsections, the different solutions are presented and discussed.

2.1.1 Esri ArcGIS

The "Georeference" tool in ArcGIS has the option to georeference a given orthophoto automatically. The process is not fully automatic, however. In order to start the process, the user must place the image to be georeferenced approximately where it is supposed to be in an already georeferenced image. This image can be much larger in extent than the image to be georeferenced. The two images are then matched (Esri, 2016b).

The tool have some limitations, which are summarized in this tip from the tool's documentation (Esri, 2016b):

> To achieve a higher success rate in [georeferencing], the two images need to be as similar as possible: geographic location, time and season, image orientation, image scale, and band combination [such as red green blue (RGB), and infrared].

Two more limitations are mentioned in the same document. One is that the images must have approximately the same spatial resolution, or that the image already georeferenced have larger spatial resolution than the other image. The second, is that the aspect ratio of the pixels must be very similar.

If one is to work with time-series, these limitations might not be a major problem.

2.1.2 Leica Geosytems IMAGINE AutoSync™

IMAGINE AutoSync™is an add-on to the software package Hexagon Geospatial ERDAS IMAGINE. It works by generating many (thousands) points for each image, and then matching these two sets of points. Very few details of the inner workings of this extraction, and matching are disclosed (Leica Geosystem Geospatial Imaging, LLC, 2005; Erdas, 2008).

Leica's solution does not have as many, and as strict limitations as Esri's; the images may be of different resolution and the bands in the images need not be identical. One image could, for example, be true color, RGB, while the other image could have a one or more bands in the infrared (IR) spectrum.

Another advantage is that the image to be georeferenced need not be placed by the user a priori (Leica Geosystem Geospatial Imaging, LLC, 2005).

Both ArcMap, and AutoSync share a common limitation; another (similar) image must already be georeferenced.

2.1.3 Leica Cyclone REGISTER

Stitching together laser scans, and can recognize targets automatically similar to Figure 2.4. 2011/

2.1.4 Li and Briggs

Unlike the previous two systems, Li and Briggs uses a georeferenced *vector* set of a road network instead of another image. This makes it possible to match an orthophoto of a constricted area, such as a building site, or a city, with the road network of the entire country in a relatively short time (minutes).

Their method finds intersections in the image, and matches these to a subset of the intersections of the road network by using TPP.

Unfortunately, their exact implementation is not publicly available. Pseudo-code of their algorithm, and their use of TPP, however, is publicly available in Li and Briggs (2006). There are not much details on how the pre-processing of the road network, and the image. As we shall see in Section 2.5.2, the concept of TPP, and the corresponding matching algorithm is well suited for solving the correspondence (matching) problem.

2.1.5 Bundle adjustment

All the previous methods rely on one dataset being georeferenced in advance. Many methods that use bundle adjustment are capable of using auxiliary information, such as the location of where the different images taken, to georeference the resulting orthophoto.

Examples of such software packages include Pix4D, AgiSoft's PhotoScan, DroneDeploy, and OpenDroneMap (AgiSoft LLC, 2012; Pix4D, 2015; DroneDeploy, 2015; OpenDroneMap, 2015)

The location of where the different images were taken can either be stored in the image itself, or in a separate file. Normally this information is gathered by using a code-based Global Positioning System (GPS) receiver, but can also be gathered with a real time kinematics (RTK) system. The latter is able to give an accuracy of the georeference comparable to manually georeference the image. However, this method gives no indication of the error, and no means to validate the result (automatically) (Nahavandchi et al., 2015).

The first option have a similar problem, but due to the use of code based GPS, and likely only a single band (L1), the base accuracy of the position is between 1 - 15 meters (Nahavandchi et al., 2015; Nahavandchi, 2015).

With any method that uses geotagged images (i.e. a image with a location associated with it), there is a problem in determining which exact pixel the geotag is associated with. One might assume that the location is associated with the center pixel. With an RTK system, one has to account for where the Global Navigation Satellite System (GNSS) receiver is in relation to the camera that took the picture. This is because that the receiver, and the camera might be more than a couple of centimeters away from each other, which is often the accuracy of a RTK system. This issue will become a greater neusence if the camera on-board a drone or aerial vehicle is facing the ground perpendicular. This might happen during some turbulence, or if the aerial vehicle is turning.

Another issue that might arise, is that the camera in the drone or aerial vehicle that took the image might not be perpendicular to the ground.

2.2 Classification

Most classifiers require some data that says what is a target, while others also need to know what is definitively not a target. Some classifiers, however, need no information of what constitutes different classes. Such classifiers are often categorized as cluster analysis. These are often used for exploratory analysis, but not often with for automatic categorization, as the algorithm do not know what constitute a certain category unless it has been "told" beforehand (Feldmann, 2015). The classifier that was used in the implementation is described in Section 2.4.1.

2.2.1 Hypothesis testing

A classifier can be seen as a, hypothesis testing. In our case, we test the hypothesis that a given pixel, or area a ground control point (GCP). The null hypothesis, then is that an area, or pixel is part of the background. For any hypothesis testing, there is a certain probability that the conclusion is false. This happens in one of two ways, Type I, and Type II errors. A Type I error is to reject the null-hypothesis when it is true, while a Type II error is to accept, or not reject, the null-hypothesis when it is false (Walpole et al., 2012; Gonzalez and Woods, 2008b).

Ideally, there would be very few Type I, and Type II errors. Unfortunately, if we want to decrease the amount of Type I errors, there will be more Type II error under the same dataset, and method to test the hypothesis (Walpole et al., 2012).

2.3 Color Theory

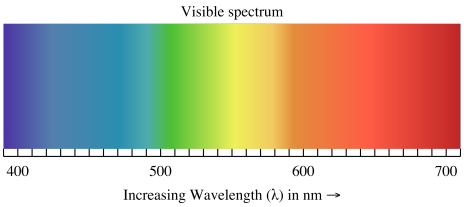


Figure 2.1: An illustration of the visible electromagnetic spectrum. Adapted from Ronan (2007).

Since the prototype of this thesis will deal with recognizing a particular color, it is fitting to review some of the theoretical framework for color.

Color, as we perceive them, is photons with a particular wavelength reflected from a surface. The wavelengths humans are capable of observing range from approximately 380 nm to 780 nm. For most people, however, the range is from 400 nm to 700 nm (Tipler and Mosca, 2008). An example of this span can be seen in Figure 2.1.

In the human eye, there are two kinds of photo sensitive cells; cones and rods. Of the first, there are three subcategories; those sensitive to red, green, and blue. Their sensitivity is approximately Gaussian, with a mean of 575 nm, 535 nm, and 445 nm respectively. All of these require ample light, or a sufficient number of photons reaching the eye per unit time. The latter is very sensitive to light, but is not able to sense color. In other words, rods are capable of detecting tiny amounts of light, but not the color of the light (Gonzalez and Woods, 2008c).

A single wavelength is not sufficient to describe colors as we observe them, however. In Figure 2.2, we see one of the reason for this. Here we see how different kinds of soil, leafs, and water reflect different wavelengths in the visible electromagnetic spectrum, and thus giving of a certain color. Additionally, some colors that we can see, are non-spectral, i.e. not a color we can see in the rainbow (Rodges, 2010; Blackwell, 2013). Such colors include pink, gold, brown, and purple. Another aspect of a color is its brightness, or luminescence, and the saturation of a color (Gonzalez and Woods, 2008d). How we can describe colors is the topic of the next section.

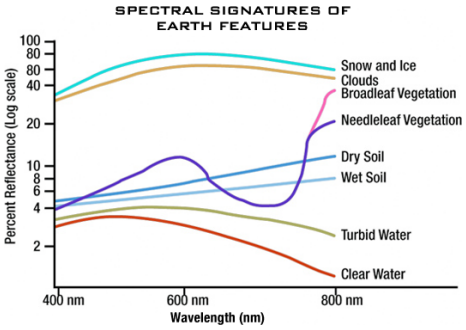


Figure 2.2: Examples of spectral signatures of different substances in the visible range of the electromagnetic spectrum. (Source: Allen (2010). Colors are inverted to better fit printing on paper.)

When an image is taken, some assumptions about the lighting condition is often required. That is, unless it is taken in a raw format. White balance is another concept that is useful when dealing with color images. The human brain is very good at determining what white looks like even when ...

2.3.1 Color models

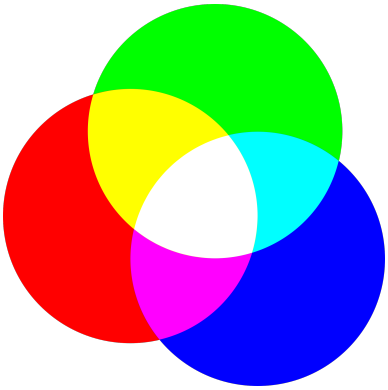
2.3.1.1 The RGB model

Since the cones of the eye is sensitive to red, green, and blue, the red green blue (RGB) is a natural color model. As the name suggests, it uses three primary colors to form other colors in an additive fashion. This model is similar to how light (photons) work; is you shine a blue light on a surface, and then shine red light on the same surface, the reflected color is magenta. For green, and blue, the result is cyan, while red and green light gives a yellow light. The RGB model is often seen as in Figure 2.3a, but is actually a three dimensional (3D) model, as can be seen in Figure 2.3b. For any color model, or vectors of color value, the different dimensions are called bands, or channels.

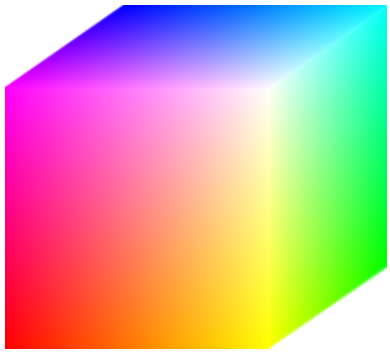
In general, exactly how the colors are encoded, and displayed varies from device to device. This makes the RGB model device dependent, and not ideal for communicating colors between applications, screens, or images. Additionally, all colors that the human eye is capable of seeing, cannot be reproduced by three static primary colors (Gonzalez and Woods, 2008d).

2.3.1.2 Hue, saturation, and value

The RGB model is very useful for dealing with additive color. The model is not intuitive, however. A more intitive description of color, is the HS* models. All of them use hue and saturation. The difference between them is in how lightness, or intensity is modeled. These models are more intuitive, as one van first select a particular hue of color (e.g. maroon,



(a) An illustration of the additive property of the RGB color model. It shows the intersection of the primary colors to become yellow, cyan, magenta, and white.



(b) An illustration of the three dimensional (3D) model of RGB. (Source: Gonzalez and Woods (2008e).)

Figure 2.3: Illustrations of the RGB color model.

teal, or blue) and then define how saturated the color is, and how much white it should have (Gonzalez and Woods, 2008d,f).

The main disadvantage for analytic purposes, is that hue is cyclic. A red color that have a hint of blue, or violet in it have a value for hue that is on the other end of the scale than pure red.

Another disadvantage is that saturation and lightness, or intensity is often strongly correlated (Kayser et al., 2008).

2.3.1.3 The Lab model

The cyclic nature of hue in HS* is unfortunate. Lab is a color model developed by the International Commission on Illumination in 1976 that does not have this problem. The model decouple color from luminance, or lightness. As the previous two model, Lab is also three dimensional. The dimensions are luminance, or lightness, a, and b. The latter two are perpendicular color vectors. a goes from red to green, while b goes from blue to yellow (Cruse, 2015). Unlike the previous two models, Lab uses absolute positioning; a given vector of Lab values is the same across multiple devices Rys (2015). The Lab color model is able to represent every color the human eye is capable of seeing in addition to colors it is not. The main advantage, and purpose of the Lab model is that (euclidean) distances between colors in the model correspond to perceived differences. That is, if two color vectors described in Lab are far apart, then they will look very different from each other. This is an advantage when looking for a particular (signal) color.

2.4 Image processing

A large part of the solution proposed in this thesis is concerned with detecting the GCPs in a given image. The reason for this, is that a large part of the prototype in Appendix A process the input image. By using TPP, and least-squares estimation (LSE), the absolute orientations of the image was found from the points extracted from the image quickly.

Most techniques known to, and used in, image processing is concerned with gray-scale, and binary images. A minority of these techniques are applicable to color images as well. Some techniques, however, only work on color images. One of the reasons for this, is that a gray-scale image is essentially an $n \times m$ matrix, while a color image can be treated as an $n \times m \times 3$ matrix.

Gonzalez and Woods (2008g) is an introductory textbook for image processing. Out of its 12 chapters, only one is dedicated to color image processing, while many of the remaining chapters are dedicated to binary images, and gray-scale images. Much of what is said here, comes from this book.

2.4.1 Segmentation

The purpose of segmenting an image, is to divide it into different regions that have different properties. In our case, we want to divide the image in two categories: "ground control point", and "background". This is why tools and techniques from image processing is used in this thesis.

GCPs can be difficult to spot. Unless they are marked with a bright signal color, we normally would not notice them in our daily lives. They are also quite small. The marked area in Figure 3.6 is 15 cm $\times$ 15 cm. With an spatial resolution of $3\frac{\text{cm}}{\text{pixel}}$, the entire GCP is 5pixel $\times$ pixel, and the GCP itself constitutes approximately a single pixel. Therefore, the GCPs must be marked in some fashion before the aerial images are taken. There are generally two ways of marking a GCP; using a signal color, as was done in Figure 3.6, or using a specific shape. Often these two are used together with an emphasis on one of them. For example, the mark can have a shape as Figure 2.4, or Figure 3.6. In the first case, the main emphasis is the shape, while in the latter, the color is the main emphasis. Image processing have methods for detecting both.

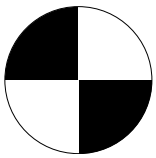


Figure 2.4: An example of a marker for a GCP.

In the orthophotos that were used in the development of the prototype, the GCPs were marked using a signal color. Therefore, this thesis will focus on techniques for detecting specific colors.

Since gray-scale images are easier to work with, it would be convenient to convert the image into a gray-scale. When an image is normally converted into gray-scale, a weighted average is used . By doing this, however, we loose much of the information of the signal color, defeating the purpose of using color markers. Assuming the target color is known,

a distance metric can be used to measure the distance of any pixel from the target instead.

One metric that will convert the color image into a gray-scale image is the Euclidean distance metric:

$$d_E(\boldsymbol{z}, \boldsymbol{a}) = \left((\boldsymbol{z} - \boldsymbol{a})^\intercal (\boldsymbol{z} - \boldsymbol{a})\right)^{\frac{1}{2}} \tag{2.1}$$

Here $\boldsymbol{z}$ is a color vector in the image, while $\boldsymbol{a}$ is the target color, also a color vector. When the target color is sampled, $\boldsymbol{a}$ can be set to be the mean of the sample data.

This metric is simple to implement, and simple to compute. If need be, the image can be normalized to the interval $[0, 1]$.

In Section 4.2.1, and 3.2, we shall see that a single measurement of the target color is insufficient to represent the color, and the variations in it. Some of this variation can be seen in Figure 3.7 and 3.6.

A metric that takes the variation of the different channels, and their interplay into account is the Mahalanobis distance metric (Kyriakidis, 2015; Wicklin, 2012; Orlov, 2011):

$$d_M(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left((\boldsymbol{z} - \boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1} (\boldsymbol{z} - \boldsymbol{\mu})\right)^{\frac{1}{2}} \tag{2.2}$$

Here $\boldsymbol{z}$ is the same as in 2.1, and $\boldsymbol{\mu}$ is the mean of the sample data (per channel) and

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11}^2 & \sigma_{21}^2 & \sigma_{31}^2 \\ \sigma_{12}^2 & \sigma_{22}^2 & \sigma_{32}^2 \\ \sigma_{13}^2 & \sigma_{23}^2 & \sigma_{33}^2 \end{pmatrix}$$

is the covariance matrix of the sample data. σ_{ij}^2 is the variance between (color) channel i and j.

The Mahalnobis distance metric can be seen has a transformation from the Euclidean n-dimensional space to an n-dimensional ellipsoidal space (Gonzalez and Woods, 2008f).

A compromise between the two, is the standardized Euclidean metric:

$$d_{sE}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \left((\boldsymbol{z} - \boldsymbol{\mu})^\intercal \operatorname{diag}(\boldsymbol{\Sigma})^{-1} (\boldsymbol{z} - \boldsymbol{\mu})\right)^{\frac{1}{2}} = \left(\sum_{i=1}^{n} \left(\frac{z_i - \mu_i}{\sigma_{ii}}\right)^2\right)^{\frac{1}{2}} \tag{2.3}$$

Here $\operatorname{diag}(\boldsymbol{\Sigma})$ is the matrix whose only non-zero entries are the diagonal of $\boldsymbol{\Sigma}$, while $\boldsymbol{z}$, and $\boldsymbol{\mu}$ are as in (2.2).

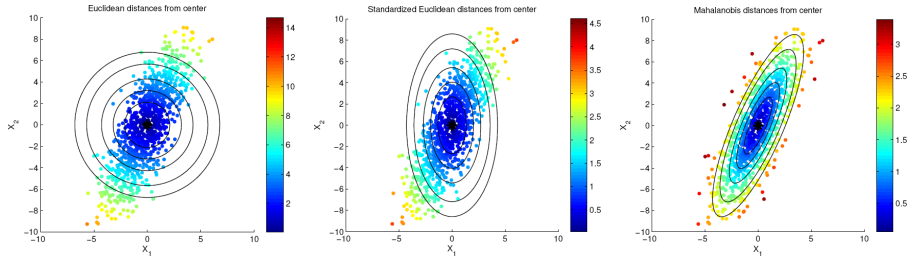
The purpose of these metrics are to convert the image into a gray-scale. The absolute distance is therefore not of interest. In order to make the computations more efficient, the square roots can be dropped, and the squaring of numbers can be replaced by an absolute value. For a prototype these optimizations are not essential, but they will improve the

running time of the program.

The metrics will be applied to matrices. In order to make the calculation more efficient (i.e. avoiding loops), the metrics can be generalized to work with matrices instead of vector, and return a vector of distances instead of a scalar. By using Script A.48 instead of the built-in function `mahal`, the time the calculation took was reduced by 99.6% for (2.2).

Figure 2.5 shows an example of how the different metrics affect the distance measurements of a dataset.

Originally, the Mahalanobis distance metric, d_M, assumes that the data is normally distributed (Mahalanobis, 1936). In Section 3.4.2, we see that the sample data is *not* normally distributed. However, this need not be a major problem, as the absolute distance is not of interest. Additionally, it is recommended for the purpose of segmenting a color image by Gonzalez and Woods.



(a) The Euclidean distance metric, $d_E(\boldsymbol{x}; \boldsymbol{\mu})$ for measuring the spread in the data.

(b) The standardized Euclidean distance metric, $d_{sE}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for measuring the spread in the data, which account for the standard deviation of each variable.

(c) The Mahalanobis distance metric, $d_M(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ which accounts for the entire covariance matrix to the sample data.

Figure 2.5: A comparison of distance metrics. The black star in the middle of the data is its mean, while the rings, and ellipses represents equidistant contour lines under the different metrics. The color indicates how far a point is from the middle. The color bar shows the magnitude the different colors represents. (Source: Kyriakidis (2015))

2.4.2 Morphology

When an image has been segmented into different regions, we need a way of determining whether a given area can be an actual GCP. Morphology means the study of shapes. One of its applications is to determine the size and shape of areas, such as the area, and eccentricity of a shape (Gonzalez and Woods, 2008h).

One particular useful operation is “hole filling”, which does what the name suggest; it finds “holes” in a shape and fills it. The hole need not be in the middle of the shape, bu can also be on the side. A shape similar to a horseshoe would be filled, and thus become more of a circle. The use of this in the context of finding GCPs might not be obvious. When a threshold is applied to the orthophoto given a certain metric, there is a chance that the

areas around the GCPs is marked as "ground control point" while the middle is not. This happened on occasion during development.

Other uses include obtaining certain parameters about each area, such as area, circumference, and eccentricity. From these parameters certain criteria can be defined that characterize a GCP. Some of these criteria are given in Script A.9.

2.5 Topological Point Pattern

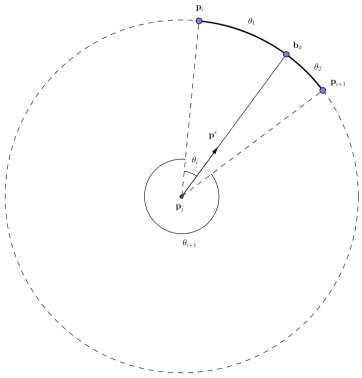


Figure 2.6: An illustration of the case of two points that have the same radius, but different angles from the principal axis. $\boldsymbol{p}_j$, is the anchor point, $\boldsymbol{p}^*$ is the point that corresponds with the principal axis, $\boldsymbol{p}_i$, and $\boldsymbol{p}_{i+1}$ are the points that are considered to be a matching for the point $\boldsymbol{b}_k$.

One of the goals of this thesis is to investigate whether topological point pattern (TPP) can be adapted to the context of matching a image to a set of measured-in GCPs. The concept and accompanying algorithm were first suggested, and developed by Li and Briggs (2006). Both are called TPP. The algorithm has been adapted by the author to better fit the context of the desired prototype.

During the implementation of TPP, some flaws with the original algorithm was discovered, and corrected. These are detailed in Section 2.5.1.1.

2.5.1 Defining topological point pattern

Given a set of (finitely many) points, say $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n\}$, where $\boldsymbol{p}_i \in \mathbb{R}^k$, a topological point pattern *relative* to the anchor point, $\boldsymbol{p}_j \in P$, is defined as the *ordered* sequence:

$$\mathfrak{T}(P_j) = \langle \boldsymbol{p}_1^* \prec \boldsymbol{p}_2^*, \prec, \ldots, \prec \boldsymbol{p}_n^* \mid \boldsymbol{p}_i^* = (r_{ij}, \theta_{ij}) \in T(P_j) \rangle \tag{2.4}$$

where $r_{ij} = ||\frac{\boldsymbol{p}_i}{\boldsymbol{p}_j}|| = \sqrt{\frac{\boldsymbol{p}_i^\intercal \cdot \boldsymbol{p}_i}{\boldsymbol{p}_j^\intercal \cdot \boldsymbol{p}_j}}$ and θ_{ij} is the counter-clockwise angle between $\boldsymbol{p}_i$ and vector defined by $\boldsymbol{p}_j - \boldsymbol{p}^*$, where $\boldsymbol{p}^*$ is the point closest to the anchor point $\boldsymbol{p}_j$. $\prec$ defines the lexicographic ordering ($\boldsymbol{p}_i = (r_i, \theta_i) \prec \boldsymbol{p}_j = (r_j, \theta_j) \iff r_i < r_j \vee r_i = r_j \wedge \theta_i < \theta_j$), and

$$T(P_j) = \left\{ \frac{\boldsymbol{p} - \boldsymbol{p}_j}{s} \mid s = \min_{\boldsymbol{p}_i \in P \setminus \boldsymbol{p}_j} d_E(\boldsymbol{p}_i, \boldsymbol{p}_j) \wedge \boldsymbol{p} \in P \right\} \tag{2.5}$$

Here d_E is the Euclidean distance metric as defined in (2.1). In words; each point is moved so that the anchor point defines the origin. All distances are then scaled, such that

the distance between the anchor point, and its closest neighbor is 1.

Li and Briggs also suggested limiting T, by introducing a maximal distance from the anchor point. This can be done by defining

$$T(P_j, d) = \{\boldsymbol{p} \mid ||\boldsymbol{p}|| \leq d \wedge \boldsymbol{p} \in T(P_j)\} \tag{2.6}$$

The TPP of a set of points is defined as a collection:

$$\mathcal{T}(P) = \{T(P_i) \mid 1 \leq i \leq |P|\} \tag{2.7}$$

In Li and Briggs (2006), k was assumed to be 2, but their algorithm can easily be extended to higher dimensions. This follows from the fact that TPP uses polar coordinates.

Since GCPs are not placed on top of each others, two dimensions should sufficient to match two sets of points.

2.5.1.1 Adjustment

The definition of $\prec$ from Li and Briggs does not work in every case. Figure 2.6 shows such a case. In this case, $\boldsymbol{p}_i \prec \boldsymbol{p}_{i+1}$, and thus $\boldsymbol{p}_i$ is considered, and tested for being a match before $\boldsymbol{p}_{i+1}$ is. It is assumed that both points are within a predefined threshold for the radius. From the illustration, it is clear that $\theta_2 = 2\pi - \theta_{i+1} < \theta_1 = \theta_i$

A better sorting scheme would then be $\prec^*$, which is defined as

$$\begin{aligned} \boldsymbol{p}_i = (r_i, \theta_i) \prec^* \boldsymbol{p}_j = (r_j, \theta_j) \iff \\ r_i < r_j \vee r_i = r_j \wedge \min(\theta_i, 2\pi - \theta_i) < \min(\theta_j, 2\pi - \theta_j) \end{aligned} \tag{2.8}$$

In other words; the points are sorted first by distance from the anchor point, then by the absolute angular distance from the principal axis.

2.5.2 Matching Topological Point Patterns

The algorithm for matching the two sets of points is not very specific in Li and Briggs (2006). Their pseudo-code says:

> SET *acm* to be the set of matching point pairs between $tpp(r)$ and $tpp(v)$.

Here $tpp(r)$ is the set of TPPs from the image, while $tpp(v)$ is the TPPs from the already georeferenced road network. On this case, however, $tpp(v)$ is equivalent to the set of TPPs formed by the GCPs.

They then go on to say that "[t]he matching between $tpp(r)$ and $tpp(v)$ is based on their sorted lists". There is little explanation beyond this other than using predefined values for $\Delta\theta$ and Δr for comparing $|r_i - r_j|$ and $|\theta_i - \theta_j|$ when the data is not perfect.

The author took this to mean that the algorithm goes through the sorted lists and compares the radii, and angles with the predefined threshold. If it is a match, it moves successive

along both lists of points. If it is *not* a match, however, some work work is required. The choice of which point to reject depend on how close the two points are, and how close their successors are. A working implementation is given in Script A.50.

The matching is mainly done by using one index for each of the two TPPs that are being matched. If the points the indices point at satisfy the Δr, and $\Delta\theta$ requirement, both indices are incremented by one. If the two points of the TPPs do not match, different cases are considered. These cases are described in Script A.50 on line $418 - 447$.

On the choice of thresholds Li and Briggs does not offer any indication of what constitutes good values for Δr and $\Delta\theta$. An empirical approach is therefore necessary. By trail and error, 0.05 was found to be suitable for both thresholds.

2.6 Absolute orientation

After a corresponding/matching between the GCP candidates extracted from the orthophoto and the measured-in GCPs have been found, the absolute orientation parameters can be computed. This section will review four methods for how this can be done. Three of these methods have been implemented in the prototype. They can be seen in Script A.13, A.14, and A.59.

In addition to these three, a fourth will be reviewed here.

Notation For all the different algorithms, it is assumed that there are two sets of n points;

$$X = \left\{\boldsymbol{p}_{x,i} = (x, y, z)^\intercal \mid 1 \leq i \leq n \in \mathbb{Z}^+ \wedge x, y, z \in \mathbb{R}\right\} \tag{2.9}$$

These are the points extracted from the image and the digital elevation model (DEM). The measured-in coordinates of the GCPs are defined similarly:

$$Y = \left\{\boldsymbol{p}_{y,i} = (x, y, z)^\intercal \mid 1 \leq i \leq n \in \mathbb{Z}^+ \wedge x, y, z \in \mathbb{R}\right\} \tag{2.10}$$

Further, it is assumed that $\boldsymbol{p}_{x,i}$ form a candidate matching (CM) with $\boldsymbol{p}_{y,i}$. That is, the image point $\boldsymbol{p}_{x,i}$ has the measured-in coordinate of $\boldsymbol{p}_{y,i}$. The mean values are defined in (2.11) for X, and (2.12) for Y.

For (2.16), (2.15), and (2.17), $\boldsymbol{p}^*_{x,i} \in X^*$ and $\boldsymbol{p}^*_{y,i} \in Y^*$.

Since both Horn (1987) and Horn et al. (1988) uses "moved" pointsets, the sets X^* and Y^* are defined in (2.13), and (2.14) respectively.

The (total) variance of the two sets are given in (2.15) and (2.16). They are primarily used to calculate the scale factor, s.

The rotation matrix is defined in (2.18) Here yaw is done first, then roll, and finally pitch. This matrix is consistent with the rotation defined in Kraus (2007).

Finally, the covariance matrix is defined in (2.17). In this matrix, S_{ij} is the covariance between the i^{th} dimension of X, and the j^{th} dimension of Y.

Conceptually, the points need not be 3D. They can be any dimension, but in practice they are usually 3D, or two dimensional (2D). If the points are 2D, many of the algorithms can be simplified. The procedure in Horn et al. (1988), however, breaks down in 2D. A remedy is therefore provided.

$$\boldsymbol{\mu}_x = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{p}_{x,i} \tag{2.11}$$

$$\boldsymbol{\mu}_y = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{p}_{y,i} \tag{2.12}$$

$$X^* = \{\boldsymbol{p}_{x,i} - \boldsymbol{\mu}_x \mid \boldsymbol{p}_{x,i} \in X\} \tag{2.13}$$

$$Y^* = \{\boldsymbol{p}_{y,i} - \boldsymbol{\mu}_x \mid \boldsymbol{p}_{y,i} \in Y\} \tag{2.14}$$

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n} ||\boldsymbol{p}_{x,i}^*||^2 \tag{2.15}$$

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n} ||\boldsymbol{p}_{y,i}^*||^2 \tag{2.16}$$

$$\begin{aligned}\boldsymbol{\Sigma}_{xy} &= \sum_{i=1}^{n} \boldsymbol{p}_{y,i}^* \boldsymbol{p}_{x,i}^{*\intercal} \\ &= \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}\end{aligned} \tag{2.17}$$

2.6.1 Least-Square error estimation

A technique that is often used when finding an optimal curve through a dataset is least-squares estimation (LSE), or fitting. This is a method of regression that finds the parameters that minimizes the sum of all squared errors of a parametric function, given a number of target values and measurements. Mathematically, it is defined in (2.19), where n is the number of measurements and expected values, $\boldsymbol{y}_i$ is the target value of the function f for the given measurements $\boldsymbol{x}$. $\mathcal{P}$ is the collection of all valid parameters to the function f, $\boldsymbol{p}$ is a particular set of parameters that makes it possible to evaluate f for the vector of measurements $\boldsymbol{c}$. An example of the latter would be two concrete values for a and b for the function $f(x) = ax + b$, such as $a = 3$, and $b = 5$, while $\mathcal{P}$, would, in this case be $\mathbb{R}^2$.

$$\boldsymbol{R}_{\theta\phi\kappa} = \begin{pmatrix} \cos\phi\sin\kappa & -\cos\phi\sin\kappa & \sin\phi \\ \cos\theta\sin\kappa + \sin\theta\sin\phi\cos\kappa & \cos\theta\cos\kappa - \sin\theta\sin\phi\sin\kappa & -\sin\theta\cos\phi \\ \sin\theta\sin\kappa - \cos\theta\sin\phi\cos\kappa & \sin\theta\cos\kappa + \cos\theta\sin\phi\sin\kappa & \cos\theta\cos\phi \end{pmatrix} \tag{2.18}$$

For this thesis, the function f is the absolute orientation. It is given by (2.20), where $\boldsymbol{x}$ is an image point in 2D or 3D, $\boldsymbol{R}$ is a rotation matrix of appropriate size, $\boldsymbol{t}$ is a translation vector of the same dimension as $\boldsymbol{x}$, and s is a scaling factor.

Theoretically, a minimum of three points are needed for 3D absolute orientation (Arun et al., 1987). For 2D, only two points are needed to solve for the four absolute orientation parameters. In practice, however, more points are wanted, and needed in order to say

something about the goodness-of-fit, and to reduce the error (Horn et al., 1988). Five or more GCPs are suggested in Skogseth and Norberg (1998c).

2.6.1.1 Kraus

$$\min_{\boldsymbol{p}\in\mathcal{P}} \sqrt{\sum_{i=1}^{n} \left(\boldsymbol{y}_i - f(\boldsymbol{x};\boldsymbol{p})\right)^2} \quad (2.19)$$

$$f(\boldsymbol{p};\boldsymbol{R},\boldsymbol{t},s) = \boldsymbol{t} + s\boldsymbol{R}\cdot\boldsymbol{p}, \quad (2.20)$$

Unlike the next three methods for estimating $\boldsymbol{R}$, $\boldsymbol{t}$, and s, Kraus suggests using an iterative approach to solving the least-square estimation problem. The starting point is to linearize (2.20), which can be expanded into (2.24).

In this equation, $[x_{\aleph}, y_{\aleph}, z_{\aleph}]^{\intercal}$ is the World coordinate of the given point $[x, y, z]^{\intercal}$. Another name for these coordinates is object coordinates and measured-in coordinates. Both points are treated as vectors in order to use linear algebra.

By using Taylor approximation, $s\boldsymbol{R}$ can be approximated by (2.25). The entire linearization of (2.24), then becomes (2.26). In these equations, ^{0} indicate an initial guess, and not the exponent, e.g. $\pi^0 = 1$.

Kraus then shows that (2.26) can be rearranged as the linear system of equations (2.21) by treating it as a least-squares estimation problem. Here v_x^i, v_y^i, and v_z^i is the residuals of the i^{th} point in x, y, and z direction. x_i^0, y_i^0, and z_i^0 are the approximate Real-World coordinates of the model, or image coordinates of the i^{th} point, and x_i, y_i, and z_i are measured-in coordinates of the respective points.

$$\begin{aligned}
v_x^i &= dx_a + x_i^0 ds + z_i^0 d\phi - y_i^0 d\kappa - (x_i - x_i^0) \\
v_y^i &= dy_a + y_i^0 ds - z_i^0 d\theta + x_i^0 d\kappa - (y_i - y_i^0) \\
v_z^i &= dz_a + z_i^0 ds + y_i^0 d\theta - x_i^0 d\phi - (z_i - z_i^0)
\end{aligned} \quad (2.21)$$

The set of equations is expanded when more points are added to the system. Note that not all points need to be 3D. Some may contain only longitude and latitude, while others might only be elevation.

This system can then be written in matrix notation as:

$$\boldsymbol{v} = \boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{l} \quad (2.22)$$

where $\boldsymbol{A}$ is the design matrix whose rows represents the set of equations from the different

points. In other words, assuming all points are known in three dimensions:

$$\boldsymbol{A} = \begin{pmatrix} 1 & & & x_1^0 & & z_1^0 & -y_1^0 \\ & 1 & & y_1^0 & -z_1^0 & & x_1^0 \\ & & 1 & z_1^0 & y_1^0 & -x_1^0 & \\ 1 & & & x_2^0 & & z_2^0 & -y_2^0 \\ & 1 & & y_2^0 & -z_2^0 & & x_2^0 \\ & & 1 & z_2^0 & y_2^0 & -x_2^0 & \\ & & & & \vdots & & \\ 1 & & & x_n^0 & & z_n^0 & -y_n^0 \\ & 1 & & y_n^0 & -z_n^0 & & x_n^0 \\ & & 1 & z_n^0 & y_n^0 & -x_n^0 & \end{pmatrix}$$

$\boldsymbol{l}$ is defined similarly for the set of $\left[(x_i - x_i^0), (y_i - y_i^0), (z_i - z_i^0)\right]^\intercal$, while $\hat{\boldsymbol{x}} = \left[d\hat{x}_a, d\hat{y}_a, d\hat{z}_a, d\hat{s}, d\hat{\theta}, d\hat{\phi}, d\hat{\kappa}\right]^\intercal$.

Kraus then shows that $\hat{\boldsymbol{x}}$ has the following solution:

$$\hat{\boldsymbol{x}} = (\boldsymbol{A}^\intercal \boldsymbol{A})^{-1} \boldsymbol{A}^\intercal \boldsymbol{l} \tag{2.23}$$

x_i^0, y_i^0, and z_i^0 are given by applying the image coordinates to (2.24).

The results from (2.23) are then applied to (2.24), and the process is repeated until the accuracy is sufficient.

$$\begin{bmatrix} x_\aleph \\ y_\aleph \\ z_\aleph \end{bmatrix} = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} + s\boldsymbol{R} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.24}$$

$$s\boldsymbol{R} \approx (s^0 + ds) d\boldsymbol{R} \tag{2.25}$$

$$= \begin{pmatrix} s^0 + ds & -d\kappa & d\phi \\ d\kappa & s^0 + ds & -d\theta \\ -d\phi & d\theta & s^0 + ds \end{pmatrix}$$

$$\boldsymbol{x} = d\boldsymbol{t} + (s^0 + ds)\boldsymbol{R}\boldsymbol{x}^0 \tag{2.26}$$

$$\boldsymbol{A}_0 = \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{pmatrix} \tag{2.27}$$

Obtaining initial parameters In order to begin the iteration, an initial guess of the seven parameters is required. Without these, initial values for x_i^0, y_i^0, and z_i^0 cannot be obtained. Consequently, neither can $\boldsymbol{A}$, nor $\boldsymbol{l}$. Often this is done manually, but since the goal of the system and prototype is to be automatic, some heuristic for the parameters is needed.

The method suggested in Kraus (2007) requires four pairs of model coordinates, and measure-in coordinates. Kraus then suggests using the points to find a similarity transform between the two sets. Ironically, then, the unknowns of (2.20) is needed to estimate the same parameters. By using exactly four points, however, the affine transformation in (2.28) can be inverted.

The inversion of (2.28) is given by solving the three equations:

$$\boldsymbol{A}_0\boldsymbol{a}_x = \boldsymbol{x}_\aleph \qquad \boldsymbol{A}_0\boldsymbol{a}_y = \boldsymbol{y}_\aleph \qquad \boldsymbol{A}_0\boldsymbol{a}_z = \boldsymbol{z}_\aleph$$

where $\boldsymbol{A}_0$ is given in (2.27). It is a matrix of the image coordinates of the four points. $\boldsymbol{a}_x = [a_{10}, a_{11}, a_{12}, a_{13}]^\intercal$, $\boldsymbol{a}_y = [a_{20}, a_{21}, a_{22}, a_{23}]^\intercal$, and $\boldsymbol{a}_z = [a_{30}, a_{31}, a_{32}, a_{33}]^\intercal$ are the unknowns, and the parameters of the affine transformation in (2.28). $\boldsymbol{x}_\aleph = \left[x_\aleph^1, x_\aleph^2, x_\aleph^3, x_\aleph^4\right]^\intercal$, $\boldsymbol{y}_\aleph = \left[y_\aleph^1, y_\aleph^2, y_\aleph^3, y_\aleph^4\right]^\intercal$, and $\boldsymbol{z}_\aleph = \left[z_\aleph^1, z_\aleph^2, z_\aleph^3, z_\aleph^4\right]^\intercal$ are vectors of the x, y, and z coordinates of the image points' respective measured-in GCPs.

These systems could be put together in one system of 12×12 equations and unknowns instead of three systems of 4×4 equations and unknowns. When computed as three systems, $\boldsymbol{A}_0$ can be inverted directly, and then applied to $\boldsymbol{x}_\aleph$, $\boldsymbol{y}_\aleph$, and $\boldsymbol{z}_\aleph$ directly.

Kraus then suggests that by comparing (2.28) and (2.20), $x_a \approx a_{10}$, $y_a \approx a_{20}$, and $z_a \approx a_{30}$, $s^2 \approx \frac{1}{3}\sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij}$. The parameters for rotation is somewhat ambiguous; $\sin\phi = r_{13}$, $\tan\theta = \frac{-r_{23}}{r_{33}}$, and $\tan\kappa = \frac{-r_{12}}{r_{11}}$. r_{13}, r_{23}, r_{33}, r_{12}, and r_{11} are obtained from (2.29). The points that constitutes this initial guess should be selected such that they are spread far from each other, otherwise one might have problems with near singularities.

$$\begin{bmatrix} x_\aleph \\ y_\aleph \\ z_\aleph \end{bmatrix} = \begin{bmatrix} a_{10} \\ a_{20} \\ a_{30} \end{bmatrix} + \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.28}$$

$$\boldsymbol{R} = \frac{1}{s}\boldsymbol{A}_0 = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{2.29}$$

2.6.1.2 Horn

Horn proposed a closed form solution to the absolute orientation problem in 1987. His solution uses quaternions to calculate the rotation matrix. Quaternions are mostly used to prove the correctness of his approach.

Quaternions One way to express rotation about an arbitrary axis is by using *quaternions* (groups of four). This concept was first introduced by Hamiltion.

A quaternion, q, is defined as

$$q = (s, x, y, z) \in \mathbb{R}^4 \tag{2.30}$$

Here, s is called the scalar part of q, while $\boldsymbol{v} = (x, y, z)$ is called the vector part. A quaterion can thus also be written as $q = (s, \boldsymbol{v})$. Another way to view quaterions is in a four dimensional complex space, where $i^2 = j^2 = k^2 = -1$ and i, j, k are perpendicular to each other Theoharis et al. (2008).

The calculation of the absolute orientation parameter The scale factor is the first parameter to be found in Horn (1987), as it is easily determined without any knowledge of the rotation of the points in relation to each other. The only information needed, is the variance for X, and Y, as defined in (2.15) and (2.16). The scale factor, s is given by

$$s = \sqrt{\frac{\sigma_x^2}{\sigma_y^2}} \tag{2.31}$$

From the covariance matrix, as defined in (2.17), a new matrix is defined:

$$\boldsymbol{N} = \begin{pmatrix} S_{xx} + S_{yy} + S_{zz} & S_{yz} - S_{zy} & S_{zx} - S_{xz} & S_{xy} - S_{yx} \\ S_{yz} - S_{zy} & S_{xx} - S_{yy} - S_{zz} & S_{xy} + S_{yx} & S_{zx} + S_{xz} \\ S_{zx} - S_{xz} & S_{xy} + S_{yx} & -S_{xx} + S_{yy} - S_{zz} & S_{yz} + S_{zy} \\ S_{xy} - S_{yx} & S_{zx} + S_{xz} & S_{yz} + S_{zy} & -S_{xx} - S_{yy} - S_{zz} \end{pmatrix}$$

The next step is to calculate the four eigenvalues, and corresponding eigenvectors.

The eigenvector, say $\boldsymbol{v}_m = (q_0, q_x, q_y, q_z)$, corresponding to the most positive eigenvalue, say λ_m, is the quaternion that represents the optimal rotation (Hamiltion, 1866).

The quaternion can be converted into a rotational matrix by using the formula:

$$\boldsymbol{R} = \begin{pmatrix} q_0^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 2(q_y q_x + q_0 q_z) & q_0^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_0 q_x) \\ 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix} \tag{2.32}$$

Finally, the translation offset can be found by calculating

$$\boldsymbol{t}_0 = \boldsymbol{\mu}_x - s\boldsymbol{R} \cdot \boldsymbol{\mu}_y \tag{2.33}$$

The eigenvalues can be found by solving the fourth order polynomial equation $\det(\boldsymbol{N} - \lambda \boldsymbol{I}) = \boldsymbol{0}$, where $\boldsymbol{I}$ is the 4×4 identity matrix. The corresponding eigenvectors can then be found by solving the equation $\boldsymbol{N}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ There are many known methods for finding eigenvalues, and eigenvectors in general, and for solving fourth order polynomials in particular. Both analytic, and numerical methods exists Abramowitz (1972); Kreyzsig et al. (2011); Lay (2012a).

The software package MATLAB offer the function `eig`, which gives the eigenvalues and the corresponding eigenvectors.

The algorithm can also be used when one wish to weight the different points differently. We then define $Y_{\boldsymbol{w}} = \{w_i \boldsymbol{p}_{y,i}\}$, and $X_{\boldsymbol{w}} = \{w_i \boldsymbol{p}_{x,i}\}$. The vector of weights, $\boldsymbol{w}$ is subject to the constraint $\sum_{i=1}^{n} w_i = 1$. One usage for this it to account for the accuracy of the different points.

This concludes the review of Horn (1987). The algorithm, and mathematics, except for weighting, has been implemented in Script A.13. In this review, quaterinons have not been prevailent because Horn use them mainly to derive the results presented here. The proof of correctness is given in Hamiltion (1866).

2.6.1.3 Horn-Hilden

The main difference between Horn et al., and Horn is how the rotational matrix is calculated. Instead of using quaterions, Horn et al. uses orthonormal matrices to calculate the rotational matrix.

It is assumed that the two point sets X, and Y are given, and consistent with (2.9), and (2.10) respectively. X is the set of image points, while Y is a set of measured-in GCPs.

$\{\boldsymbol{p}_{x,i}\}$, and $\{\boldsymbol{p}_{y,i}\}$ denote the measured-in GCPs, and the image coordinates respectively. As before we define $\boldsymbol{p}^*_{x,i} = \boldsymbol{p}_{x,i} - \boldsymbol{\mu}_{x,i}$, and $\boldsymbol{p}^*_{y,i} = \boldsymbol{p}_{y,i} - \boldsymbol{\mu}_{y,i}$, where $\boldsymbol{\mu}_x = \sum_{i=1}^{n} \boldsymbol{p}_{x,i}$, and $\boldsymbol{\mu}_x = \sum_{i=1}^{n} \boldsymbol{p}_{y,i}$. The scale factor, s, is identical to (2.31), and the transnational offset, $\boldsymbol{t}_0$, is the same as (2.33). Instead of (2.17) its transpose, $\boldsymbol{M} = \Sigma_{xy}^{\intercal} = \sum_{i=1}^{n} \boldsymbol{p}^*_{y,i} \boldsymbol{p}^{*\intercal}_{x,i}$, is used.

The main emphasis of Horn et al. (1988) is to show that

$$\boldsymbol{R} = \boldsymbol{M} \left(\boldsymbol{M}^T \boldsymbol{M} \right)^{-\frac{1}{2}} \tag{2.34}$$

Horn et al. then show that this matrix is orthonormal.

This method is not as general as Horn (1987), as this particular solution does not work when one, or both of the sets X and Y are co-planar i.e. 2D. However, a remedy is provided:

$$\boldsymbol{R} = \boldsymbol{M}\boldsymbol{S}^{+} \pm \boldsymbol{u}_3\boldsymbol{v}_3^{\intercal}, \tag{2.35}$$

where λ_1, and λ_2 are the eigenvalues of $\boldsymbol{M}^{\intercal}\boldsymbol{M}$. $\boldsymbol{u}_1$, and $\boldsymbol{u}_2$ are the eigenvectors that correspond to λ_1, and λ_2 respectively,

$$\boldsymbol{S}^{+} = \frac{1}{\sqrt{\lambda_1}}\boldsymbol{u}_1\boldsymbol{u}_1^{\intercal} + \frac{1}{\sqrt{\lambda_2}}\boldsymbol{u}_2\boldsymbol{u}_2^{\intercal},$$

where $\boldsymbol{u}_3$, and $\boldsymbol{v}_3$ are the third column in the matrices $\boldsymbol{U}_0$, and $\boldsymbol{V}_0$ respectively. These two matrices are defined by the singular value decomposition of $\boldsymbol{M}\boldsymbol{S}^{+}$ into $\boldsymbol{U}_0\Sigma_0\boldsymbol{V}_0^{\intercal}$. The sign of $\boldsymbol{u}_3\boldsymbol{v}_3^{\intercal}$ is chosen such that the determinant of $\boldsymbol{R}$ is positive.

Horn et al. (1988) has been implemented is Script A.14.

2.6.1.4 Umeyama

Umeyama focuses primarily on computer vision application. One of the reasons this article was written, was to address a problem the previous two solutions have; they may give a reflection instead of a rotation if the data is severely corrupted (Umeyama, 1991; Horn et al., 1988). The method is implemented in Script A.59.

This particular algorithm works by computing the singular value decomposition of the covariance matrix to find the rotation matrix.

The notation from the article has been changed to be consistent with the previous sections.

Sigular value decomposition A particular method for factorizing *any* $n \times m$ matrix, say $\boldsymbol{A}$ into, the matrices $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\intercal}$, is sigular value decomposition (SVD) (Lay, 2012b). Here $\boldsymbol{\Sigma}$ is a semi-diagonal matrix having the same size as $\boldsymbol{A}$. $\boldsymbol{\Sigma}$ consist of the first r singular values of $\boldsymbol{A}^{\intercal}\boldsymbol{A}$:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 & 0 \\ 0 & \ddots & & & \vdots \\ & & \sigma_r & & \\ \vdots & \dots & & 0 & 0 \\ 0 & \dots & \dots & & 0 \end{pmatrix}$$

The singular values, σ_i for $1 \leq i \leq r$ are the square roots of the eigenvalues of $\boldsymbol{A}^{\intercal}\boldsymbol{A}$ in descending order. r is the rank of $\boldsymbol{A}$. $\boldsymbol{V}$ is a matrix of the (normalized) eigenvectors of $\boldsymbol{A}^{\intercal}\boldsymbol{A}$. Their order is according to the corresponding eigenvalues. For example $\boldsymbol{V} = \begin{pmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \end{pmatrix}$.

Finally, $\boldsymbol{U}$ is constructed from $\frac{\boldsymbol{Av}_i}{||\boldsymbol{Av}_i||}$ for $1 \leq i \leq r$ in order of eigenvalues corresponding to the eigenvectors (Lay, 2012b).

Both $\boldsymbol{U}$, and $\boldsymbol{V}$ are orthonormal. Consequently, $\boldsymbol{UU}^\intercal = \boldsymbol{U}^\intercal\boldsymbol{U} = \boldsymbol{I} = \boldsymbol{VV}^\intercal = \boldsymbol{V}^\intercal\boldsymbol{V}$. In other words, $\boldsymbol{U}^\intercal = \boldsymbol{U}^{-1}$, and $\boldsymbol{V}^\intercal = \boldsymbol{V}^{-1}$ (Lay, 2012c).

Eigenvalues are the solutions to the equation $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = \boldsymbol{0}$. Eigenvectors are then computed from solving $\boldsymbol{Ax} = \lambda_i\boldsymbol{x}$, where λ_i are the eigenvalues (Lay, 2012b).

The calculation of the absolute orientation parameters Umeyama's solution is a refinement of the solutions presented in Horn et al. (1988) and Arun et al. (1987).

The article uses different notation from the previous two articles, and from the convention set forth in this thesis. It has been changed to fit.

The method of calculating the absolute orientation parameters presented in Umeyama (1991) also uses X in (2.9) for the points in the image, and Y for the measured-in GCPs in (2.10).

The covariance matrix, $\boldsymbol{\Sigma}_{xy}$, in (2.17) is divided by n and then decomposed into $\boldsymbol{UDV}^\intercal$. The decomposition is by SVD (as described above).

In order to calculate the rotational matrix, Umeyama creates the matrix

$$\boldsymbol{S} = \begin{cases} \boldsymbol{I} & \text{if } \det(\boldsymbol{U})\det(\boldsymbol{V}) = 1 \\ \text{diag}(1, 1, \ldots, 1, -1) & \text{if } \det(\boldsymbol{U})\det(\boldsymbol{V}) = -1 \end{cases} \tag{2.36}$$

The purpose of $\boldsymbol{S}$ is to ensure that $\boldsymbol{R}$ has a positive determinant. $\boldsymbol{S}$ has the same dimensions as $\boldsymbol{U}$ and $\boldsymbol{V}$.

The rotational matrix, the scale, and the translational offset are then given respectively as:

$$\boldsymbol{R} = \boldsymbol{USV}^\intercal \tag{2.37}$$

$$s = \frac{1}{\sigma_x^2}\,\text{tr}(\boldsymbol{DS}) \tag{2.38}$$

$$\boldsymbol{t}_0 = \boldsymbol{\mu}_y - s\boldsymbol{R}\boldsymbol{\mu}_x \tag{2.39}$$

Here σ_x^2 is given by (2.15), $\boldsymbol{\mu}_y$ by (2.12), and $\boldsymbol{\mu}_x$ is given by (2.11). tr is the trace of the matrix, or the sum of the diagonal.

In his article, Umeyama claims that (2.38) obviously minimizes the total error. His claim was checked, and found to be false. His algorithm is implemented in Script A.59. This gave much worse results than by using (2.31) instead of (2.38). Changing the scale factor to the same as in Horn (1987) and keeping everything else the same, the root mean square error (RMSE) of the absolute orientation parameters was improved. Section 5.7 and 5.1.2 explains this in more detail.

2.7 Licensing

One of the goals of the thesis is to publish the prototype as open source, for anyone to use has they find convenient or useful. There are many licenses to choose from, all from the "unlicense", which waives all rights of the creator, to the GNU General License, which forces anyone who use the source code to also publish it as open source, and under the same license. This is called "copyleft", instead of "copyright".

The "copyleft" licenses prohibit some of the uses, such as including it into a proprietary system. A compromise between these two extremes, are licenses such as Mozilla Public License, which grants the usage of the entire code base, or parts of it for *any* purpose, as long as any modifications to the files of the code base is made public. This is the most important reason for the source code in Appendix A is licensed under the Mozilla Public License Version 2.0 (MPL 2.0). The license is given in full in Appendix C.

Chapter 3

Method

NOW THAT the theoretical framework is laid down for a program that can accomplish the goals set forth in Section 1.1. The design process and the design of the program itself is presented in this chapter.

Along with this, the experimental setup is described.

We first start out with a minimal shell of a design, which is "fleshed-out" as we go along. The full prototype, with every detail for a functioning computer program is given in Appendix A.

3.1 The design of the program

Any implementation of the goals in Section 1.1 will need to accomplish these three tasks:

(α) Detect the ground control points (GCPs) in a given orthophoto, or aerial image.

(β) Compute a set of candidate matchings (CMs).

(γ) Calculate the absolute orientation parameters.

These steps need not be done sequentially. It might be possible to tweak bundle adjustment to accomplish them simultaneously. Since the goal is to *investigate* the possibility, and feasibility of such a system, simplicity was chosen over power, and complexity of bundle adjustment and similar methods.

Figure 3.2 shows a flow chart of what an implementation might look like. This design forms the basis of the prototype. Each process is described in greater detail throughout this chapter.

This design has some flexibility built-in; it can give two dimensional (2D) or three dimensional (3D) absolute orientation parameters. These can then be given directly, as four, or

seven parameters respectively, as a transformation matrix, written to an auxiliary image-file, or be written directly in the image.

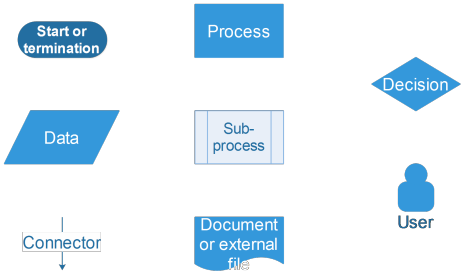


Figure 3.1: Legend for the flow charts.

As a consequence of the flexibility, the input "digital elevation model" is optional in this design. The two other inputs, "Orthophoto", and "ground control point (GCP)" are required.

3.1.1 Description of the steps

Now we take a closer look at the different steps involved in Figure 3.2.

3.1.1.1 Input parameters

The input "Orthophoto" need not be a true orthophoto, but can be any image, as long as the GCPs are visible. If the image is not a orthophoto, however, inaccuracies in the absolute orientation are likely to occur due to the use of central projection.

"Ground control point (GCP)" is a list of the measured-in coordinates of the GCPs. In the implementation, this can be either a matrix of points, or a GeoJSON file. The orthophoto will get the same projection as the GCPs, so an explicit projection is not necessary. If the projection is known, or included, it can be written to a file, or in the image for external programs to use.

"digital elevation model " was chosen to be an optional input because a 3D absolute orientation is not always needed, or available. An example is when using spatial analysis to look for a geographic pattern. In these cases, it is often assumed that the features of interest lies on the ground.

3.1.1.2 Finding GCP candidates in image

This step is essentially a classifier. It classifies pixels as either "background", or "ground control point". When every pixel have been classified, areas that satisfy certain extra criteria are extracted, along with their corresponding elevation of that area (if available). These criteria include restrictions on size and shape. The centers of these areas are then calculated, and returned as a set of GCP candidates. Areas that satisfy all constrictions are then reduced to a single point. This point may be the centroid of the area, or it can be chosen by a utility function.

In Figure 3.2, no assumptions has been made about what kind of classifier this step is. As we saw in Section 2.2, we cannot assume that "Find GCP candidates in image" finds all the GCPs. Nor can we assume that it *only* found GCPs.

Figure 3.3, shows how this problem can be solved. Assuming that more than three GCPs were found and matched initially, this design uses the inverse of the absolute orientation parameters to locate where the remaining GCPs should be in the given orthophoto. An area around these points is then extracted. More precise methods are then used for these areas. There are many approches to deciding the size of the area extracted. One method

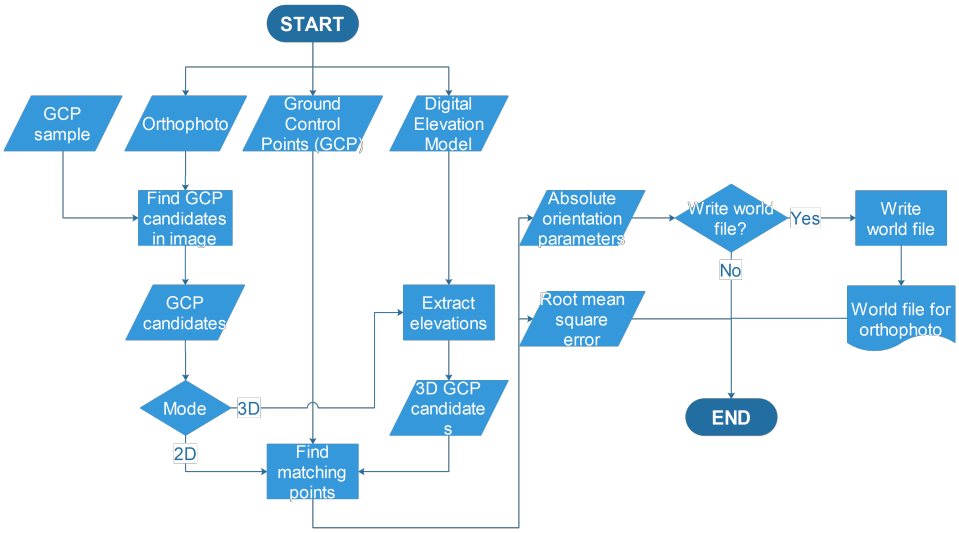


Figure 3.2: A flow chart of how a relatively simple implementation of the goals in Section 1.1 might look like. The symbols are explained in Figure 3.1.

is to make the area proportional with the root mean square error (RMSE). Another is to extract a given proportion of the entire orthophoto. One could have used more precise methods for finding the GCPs initially, but that would take much more time, which goes against the goal of the prototype being feasible. Another reason for this approach is that by examining an area around where a GCP should be, the pixel with the most utility could be chosen when the GCPs are not easily seen.

In the beginning of the development, more precise methods were used for the entire image. However, doing so made early prototypes run over-night for results, instead of a few minutes, which later versions are able to.

Extracting elevations In order to get a 3D absolute orientation, the elevation of each image point is needed. The height of each measured-in GCP is also needed.

In this step, the elevation of each area that was classified as “ground control point” is extracted, and averaged. The elevations are then appended to the centroid of the respective area as before. This is then returned as “3D GCP candidates”.

3.1.1.3 Matching points

Now we have two sets of unordered points. Ahead of time, it is not know whether the set “GCP candidates” contains all the GCPs in the image, nor if all the candidates are GCPs. In other words, it is not know whether the correspondence between “GCP candidates” and “ground control point (GCP)” is one-to-one, onto, both, or neither.

If the distances are computed, this problem can be solved as an instance of Subgraph isomorphism. This problem is known to be NP-Complete, however. Unfortunately, there

are no known algorithm to these problems in polynomial time. That is, the time it takes to run is a polynomial function of the size of the input, say number of pixels in the image, and the number of GCPs (Cormen et al., 2009).

By taking advantage of the structure of our problem, the subgraph isomorphism problem can be avoided completely. Section 2.5 introduced topological point pattern (TPP), which can be used to match the two sets of points, even when the relation between the two sets is unknown (i.e. whether the correspondence is one-to-one, onto, both, or neither). This is accomplished by taking advantage of the pattern that the two sets of points form.

When a matching has been found, the image points and corresponding measured-in coordinates are put together into a candidate matching (CM).

3.1.1.4 Finding the absolute orientation

Now that we know the measured-in coordinate of each GCP in the image, we can calculate the absolute orientation parameters. There are both iterative, and closed form solutions to this problem. Since one of the goals of the system is to be completely automatic, a closed form solution is preferred, as it does not require an initial guess. Some closed form solutions include Arun et al. (1987), Horn (1987), and Umeyama (1991).

Step (β), and (γ) are sequential, but are shown to be done simultaneously in Figure 3.2 for compactness.

3.1.1.5 Generating the output

Now that the rotation, translation, and scale factor for the orthophoto is found, they can be exported. Mathematically, the parameters are $\boldsymbol{R}$, $\boldsymbol{t}$, and s respectively.

If the image is a Tagged Image File Format (TIFF), the transformation parameters can be written as predefined tags directly inside the image. Alternatively, the data can be stored as a separate world file. Such a file specifies the location, rotation, and spatial resolution, or scale, of the image in a certain coordinate system. The world files format is quite old, but the file is still in use by ArcGIS, and PhotoScan, amongst others (Esri, 2009; AgiSoft LLC, 2012). It was first specified by Esri sometime before 1995 (Environmental Systems Research Institute. Redlands, 1995; Esri, 2016a).

Unfortunately, the file format does not support an explicit coordinate system. Instead, it must either be defined by the user when importing the orthophoto into another program, or stored as a tag in the image as Exchangeable Image File Format (EXIF). This can only be done with Joint Photographics Experts Groups (JPEG), and TIFF images, however.

Another option, is to simply print the absolute orientation parameter to the screen for the user to see.

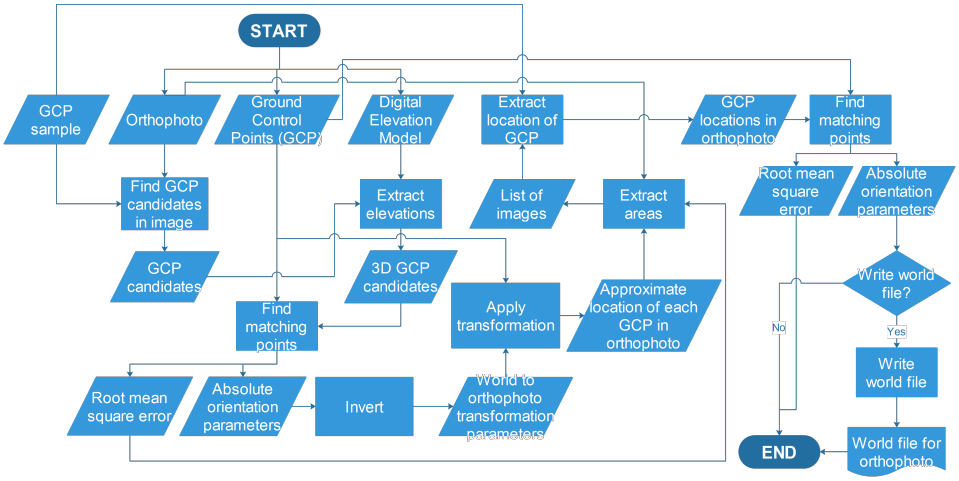


Figure 3.3: A flow char of an implemenatation that uses Figure 3.2, but uses the GCPs twice. The second time, they are used to get the image location of the GCPs given than the initial absolute orientation is not too far off. The symbols are explained in Figure 3.1.

3.2 The development of the program

This section covers how the prototype was developed.

The development, testing, and experiment was done on the author's private computer; "Tøffen". It is desktop with a quad core Intel i7-2600K, 16 GB of DDR3 memory, and an Nvidia GTX 970 graphics card running Windows 10. Both the processor, and the graphics card where over-clocked.

3.2.1 Choosing a Programming Language

A working prototype must be written in some programming language. There are many options; C/C++, C#, D, Fortran, Python, Matlab, Java, JavaScript, Haskel, Lisp, and many more.

From the flow chars in Figure 3.2, and 3.3, we see that the program is mostly procedural, or sequential. Therefore, the chosen language need not have a strong focus on object orientated practice.

A large community, and a large collection of relevant external libraries are both desired. Since the main goal of this thesis is to investigate the *possibility*, and *feasibility* of a system capable of automatic georeferencing an orthophoto, it would be convenient to reuse existing technology as much as possible. If the methods, and algorithms of image processing had to be implemented from scratch in the chosen language, much time and energy would go to waste. This is one of the reasons the language D was not chosen, even though the author is quite fond of it.

Since the program will deal with large images, some compromise between the ease of use

and prototyping of interpreted languages, and the efficiency of compiled languages must be made.

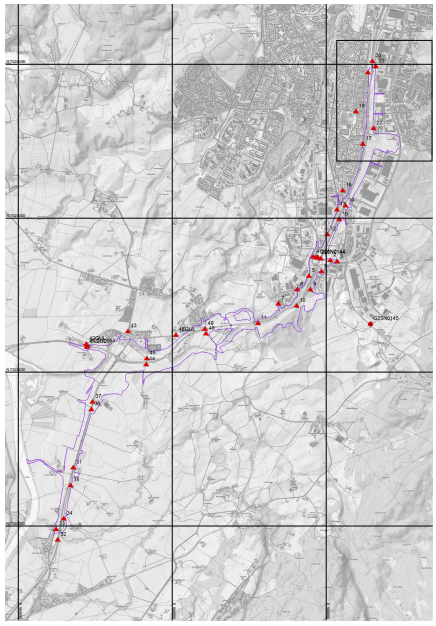


Figure 3.4: An overview of the different ground control points that were established, and measured in for the "E6" dataset. The rectangle in the upper right corner indicates where the dataset called "E6" is from. From top to bottom the GCPs are called "20", "21", "19", "18", "22", and "17". The measured in coordinates of these are given in Table B.2, where they are marked with *.

The author is well versed in Python, Matlab, and Java, and have experience with C/C++, D, and JavaScript. C/C++, and D are compiled languages, and can thus be expected to have a much shorter runtime when the program is executed. Python, Matlab, and JavaScript are all interpreted languages. This makes is easy to prototype, as one need only write the code, and then run it with a given input. Java falls in between, and is compiled to byte code, which lies closer to machine code than human-readable code, but is interpreted by a byte code interpreter at runtime.

Except for D and JavaScript, all the languages have extensive libraries for image processing. Those that exist, lack more advanced features such as morphology (AntonAL, 2015; Ludwig et al., 2016).

Both Python and Matlab are capable of running C/C++ code that have been compiled. Thus they can both take advantage of pre-made, compiled libraries, and get the same efficiency as compiled languages when functions in the library are called directly.

OpenCV is such a library. It implements many algorithms for image processing in general, and computer vision in particular. The library can be used by any language, as long as there are bindings to it.

In combination with Python, and some of Python's other libraries for heavy computation (e.g. NumPy), one can have many of the advantages of compiled languages, while retaining the ease of prototyping that Python offer. Unfortunately, NumPy and OpenCV have major problems in handling images that are larger than approximately 4GB (Nistad et al., 2016). This manifested itself in the inability to load a large image into memory.

Due to the author's unfamiliarity with OpenCV, and NumPy, in addition to time constraint of this thesis, Matlab was chosen instead. Additionally, OpenCV had problems of opening images that had a size larger than approximately 4 GB (Nistad et al., 2016). The choice was also motivated by Matlab being very efficient in dealing with matrices. In Section 2.4, we saw how an image can be treated as a matrix. Additionally, Matlab has an Image

Processing Toolbox, which implements most algorithms known in the image processing literature (MathWorks, 2016).

As a student at Norwegian University of Science and Technology (NTNU), the Author has access to Matlab, and all Toolboxes, which removes the obstacle of cost of purchase. Additionally, engineering students at NTNU receives training in the use of Matlab.

Unfortunately, Matlab is proprietary, and the cost can be prohibiting. The standard version costs 17 500 NOK for commercial use, while the Image Processing Toolbox, and Statistics and Machine Learning Toolbox costs 9 000 NOK each. This is unfortunate, as one of the goals is to make the program completely open source.

3.2.2 Test-driven development

The main reason for choosing an interpreted programming language was the ease of prototyping. The program does not have to be recompiled when new functionality is written, or when a small change to a single file of source code is made. This saves time, and makes it easier to do test-driven development.

Many of the early prototypes of the program consisted mostly of commands to see how the GCPs could be easily detected. In these instances, it becomes cumbersome, and impractical to use compiled languages.

The main approach to the development of the prototype was to develop a hypothesis for how the marked GCPs could be detected, based on the literature of image processing. The hypothesis was then tested against the extracted GCPs from the orthophoto "Lerkendal" seen in Figure 3.7. If the method the hypothesis formulated worked sufficiently well on Figure 3.7, it was applied to an excerpt of the orthophoto. If the method was able to georeference the excerpt, it was applied to the entire orthophoto. After the hypothesized method would georeference "Lerkendal" consistently and accurately, the method would be applied to a second orthophoto; "E6", which the prototype would not have seen before.

The use of a second orthophoto is validation testing. The entire development of the prototype in this thesis can be seen as a form of supervised learning (Kyriakidis, 2015). In supervised learning, and machine learning in general, it is common to divide a dataset into two, or three subsets; a training set, a test set, and some times a validation set. In this case, Figure 3.7 and excerpts from the orthophoto "Lerkendal" can be seen as training data, while the entire orthophoto can be seen as the test set. The orthophoto "E6", then is the validation set; as it is not involved in developing the prototype directly, only to validate it.

3.3 On the method of testing and verification

Different experiments are to be run with a working prototype. The first set is to use the orthophoto "Lerkendall" along with the corresponding digital elevation model (DEM) and the reference colors described in Section 3.4.2.1. The algorithm for calculating the absolute orientation parameters is then varied. The algorithms to be used are Horn, Horn-Hilden, Umeyama, and Umeyama*. This to see if the choice of algorithm can be made arbitrarily. The difference between Umeyama and Umeyama* is that the fist is a direct implementation from Umeyama (1991), while the latter uses the scale factor defined in Horn

(1987) instead of the original. The reason for this is to check the validity of the claim made in Umeyama (1991), that (2.38) *obviously* gives the optimal absolute orientation.

The orthophoto "E6" will be used to check how the prototype fares against a dataset it has not encountered during development, and to see how it deals with GCPs that are considerable smaller than those in "Lerkendal".

3.4 Acquiring data

Two datasets were used for the experiment, and testing of the prototype. Both consist of a set of images taken by a drone from FlySense called eBee. The first set was taken over Lerkendal in Trodheim, Norway. (WGS84: $63.414°$N, $10.408°$E) It consists of 448 images. These images covers a an area of approximately 0.1112 km^2. They were taken on the 13^{th} of October 2014 with a Canon PowerShot ELPH 110 HS at 4608 x 3456 pixels (15.9 megapixels). NTNU-Geomatics is the owner of these images. An overview of the area can be seen in Figure 3.5, and 3.5.

The second was taken along the highway E6 south of Trondheim, close to Tiller and Heimdal (WGS84: $63.352°$N, $10.369°$E). In total, this dataset consists of 1913 images, but only 197 were used to create an orthophoto. These image were taken on the 26^{th} of November 2015 by the same camera. They are owned by Norwegian Public Roads Administration (NPRA).

The reason for limiting the number of images used in the creation of the orthophoto "E6" is that the resulting orthophoto from using all the images was approximately 21 GB, and close to 91 GB when loaded into memory and converted from 8 bit to double. Additionally, much of of the image was empty, i.e. black. The set of images consisted of five parts. Each part was of a particular section of the highway, was was taken in a single flight. The data from the first flight was then chosen to represent "E6". One of the reasons for choosing this part is that it had a good spread of GCPs. Additionally it was not too large so that the prototype would not be able to run on a normal consumer desktop computer.

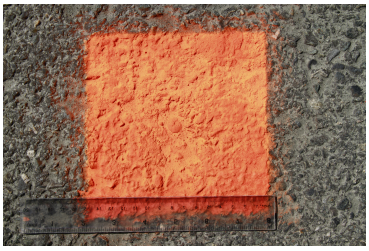


Figure 3.6: A closeup of what a ground control point looks like. It is a square with sides of 15 cm. The color is called "flashing orange"

Trond Arve Haakonsen established, measured in, and post-processed the GCPs for both datasets. In post-processing, the points were averaged, corrected, and equalized(?) by using... Nahavandchi et al. (2015) describes how the GCPs were measured and established for "Lerkendal". Table B.1 shows the measured-in coordinates of the GCPs used with "Lerkendal". The GCPs used in "E6" are shown in Table B.2. Only the points marked with an asterisk (*) can be seen the the orthophoto that was produced.

The location of the different GCPs can be seen in Figure 3.5 and 3.5 for "Lerkendal". A similar overview for "E6" is given in Figure 3.4 and 3.12.

The first dataset, and the corresponding orthophoto will be referred to as "Lerkendal", or

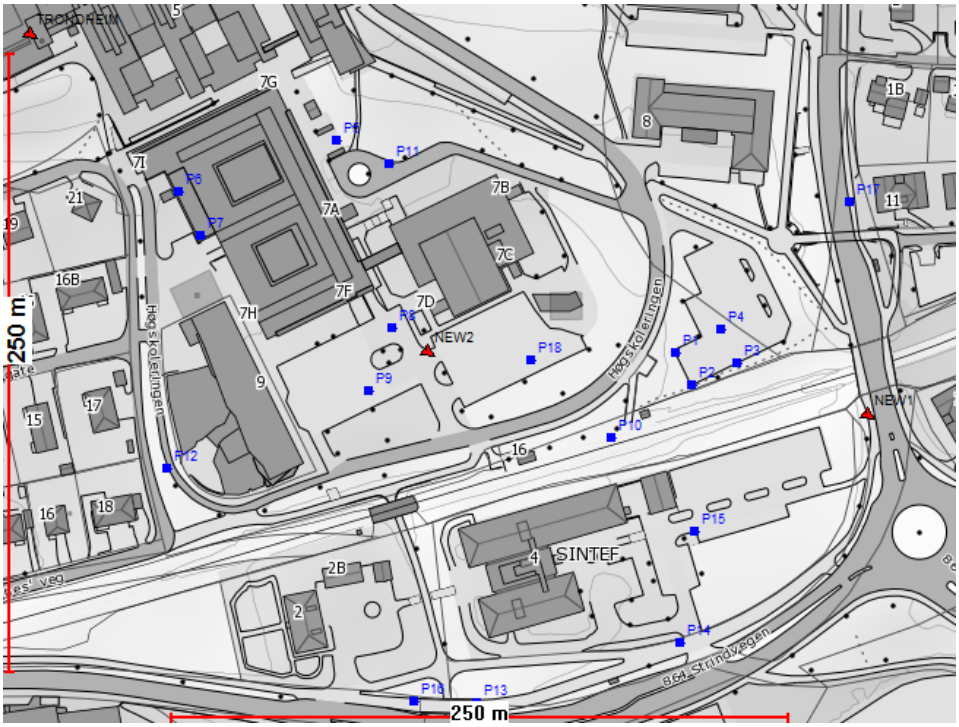


Figure 3.5: An overview of where the different GCPs are located in the dataset "Lerkendal". A closeup of the different points can be seen in Figure 3.7, while the measured-in coordinates can be seen in Table B.1. (Source: Nahavandchi et al. (2015))

the dataset “Lerkendal”, while the latter will be referred t as “E6”, or the dataset “E6”.

3.4.1 Processing the images

AgiSoft’s PhotoScan Professional Edition (version 1.2.4 build 2399 (64 bit)) was used to convert the two sets of images into two orthophotos. The highest possible settings for quality was used in all steps of the process.

The entire process was run as a batch process. The steps that were carried out was “Align Photos” with the parameter “Accuracy” set to “Highest”. The remaining parameters were set to their default value. The process “Optimize Alignment” was then run with the default values of all the parameters. “Build Dense Cloud” was the next process. “Quality” was set to “Ultra high”, and “Depth filtering” was set to “mild”. The remaining parameters had their default values. Next in line was the process “Build Mesh”. The only change in the parameters was “Surface type”, which was set to “Height field”. The next processes were “Build Texture”, “Build DEM”, and “Build Orthomosaic”. All of these processes used their default values. Finally, the DEM and orthophoto were exported to disk as single images.

Figure 3.7: A closeup of all the GCPs in the dataset “Lerkendal”. From the top left corner to the right, the name of the GCPs are “New 1”, “New 2”, “P1”, “P2”, “P3”. The second row from the top has the points “P4”, “P5”, “P6”, “P7”, and “P8”. The third row; “P9”, “P10” “P11”, “P12”, and “P13”. Finally, the fourth row shows “P14”, “P15”, “P16”, “P17”, and “P18”. The location of each of these points are shown in Figure 3.5. The GCP is in the middle of each “subimage”.

The orthophoto “Lerkendal” is $20\ 051 \times 22\ 039$ pixels, while “E6” is $47\ 619 \times 15\ 828$ pixels. The size of the marked GCPs is approximately 20 pixels across for “Lerkendal”, while “E6”’s GCPs are approximately 10 pixels across. These numbers were obtained by zooming down to a level where every pixel is visible, and counting the pixels that constitute the visible GCP. In other words, these numbers are effective size of the GCPs.

This process took about 24 hours per set of images. Both sets of images were sent through the same process.

All this processing was done at a photogrmetry workstation at NTNU-Geomatics. The computer has two Intel Xeon E2-2670 v2 @ 2.5 GHz, 32 GB of DDR3 memory (RAM), an Nvidia Quadro K5000 graphics card with 4 GB dedicated DDR5 RAM, 1 TB of Solid State Disk secondary storage, and 3 TB of hard drive. The computer runs Windows 10 Educational.

3.4.2 The sample of GCP

As described in Section 2.4.1, the Mahalanobis distance metric must have a reference sample to calculate distance from. Ideally the mean, and covariance matrix would be

found analytically from the reference color. Unfortunately this was not possible. The retailer (Blinken AS) of the paint that was used to mark the different GCPs does not have that information.

Two approaches for obtaining sample data of the color are described next.

3.4.2.1 Directly from the orthophoto

After the orthophoto "Lerkendal" was produced, the image coordinates of the GCPs were found by manually selecting them with the "Data cursor" tool in MATLAB. Script A.10 was then used to extract an area of 201×201 pixels from the orthophoto "Lerkendal". These areas where then fused together into a single image by using Script A.66. Then the application "Color Thresholder" in MATLAB was used to create a binary image around the GCPs. This was done by circling a selection of the GCPs, and then clicking on "Find Thresholds". The color mode for the application was set to Lab, since the dimensions are less correlated than red green blue (RGB). The resulting binary image was then fed into Script A.56 for removal of areas too small, and too eccentric. Finally, the binary image and the fused GCPs are fed into Script A.65 which gives an $n \times 3$ matrix of sample data.

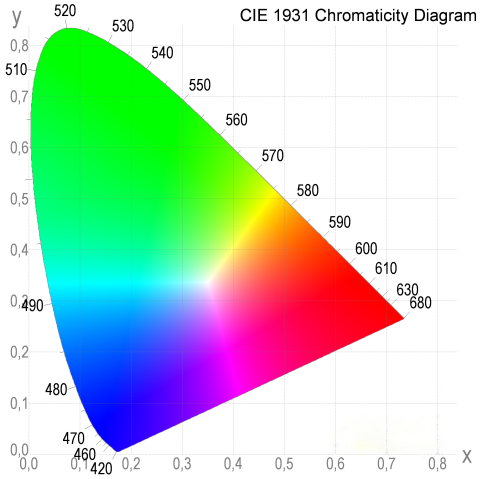


Figure 3.8: The CIE 1931 chromaticity diagram. The numbers along the edge is the wavelength of that color in nanometers. (Source: Glynn (2009). Background has inverted color to better fit printing on paper.)

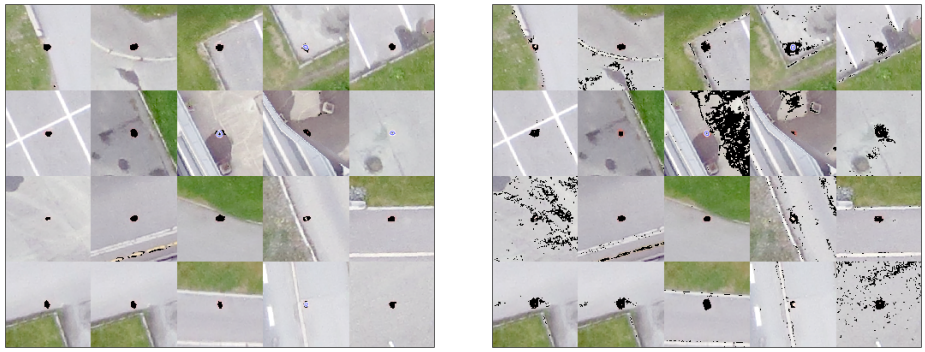
3.4.2.2 Capturing a marked GCP

Since the signal color the GCPs are marked with is not available as specifications, one might be tempted to simply take a picture of a marked GCP. For accurate values of the color, this is not as simple as it sounds. One of the reasons for this is white-balance. Another reason is chromatic aberration. Both are described, and dealt with in Section 2.3.

Figure 3.10: The JPEG version of the image that was used to create sample data from marked GCPs.

Since the images over "Lerkendal" were taken two years ago, and they have not been repainted for a year, Terje Skogseth painted the point "NEW2" anew. This can be seen in Figure 3.6. It has the same dimensions as all the other GCPs; 15×15 cm.

Figure 3.6 and similar images were captured by the author using a Canon 7D with a Sigma 17-77 mm F2.8-4 DC MACRO OS HSM lens. The images that were used to extract the color value were taken in RAW-format @ 5184×3456 pixels (17.9 megapixels), a F-stop of f/7.1, ISO-100, exposure time of $1/400$ seconds, and at a focal length of 70 mm. The



(a) The color model for thresholding is Lab. **(b)** The color model for thresholding is RGB.

Figure 3.9: The mask of Figure 3.7 after three representative GCPs were selected (blue circles). In (a) and (b) the same GCPs were selected, but due to the strong correlation between the different bands of RGB compared to Lab, many uninteresting areas where also selected in (b). The mosaic of GCPs is the same as Figure 3.7.

color profile Adobe RGB (1998) was used to define the RGB space. The profile captures a greater portion of the chromatisity diagram in Figure 3.8 than sRGB, which is normally used in screens, and cameras (Adobe Systems Inc., 2005). Each band was sampled at 14 bits resolution. With RAW, white-balance can be set in post-processing, along with any color settings.

Post processing The post processing consists of two steps; "developing the negative", and converting it into a set of samples. The first part was done in Camera Raw 9.5.1 for Adobe PhotoShop CC (2015). Since the image was taken in the RAW format, the image have to be processed in order to use them in any other context (Cairns, 2013).

In post processing, the color profile was set to "Camera Neutral" in order for no color to be emphasized. Lens distortion, and chromatic abbreviation was set to be removed automatically. White balance was set to "daylight" because the image was taken around noon in summer with a clear sky. The exposure, contrast, highlight, shadows, whites, and blacks where set automatically, and then adjusted such that non of the data captured by the camera sensor is cut out when transformed into 8 bits. In other words, no channel is overexposed, nor underexposed, and stretches across the entire range of allowed colors.

In practice, this was done by setting giving the value of -0.47 to exposure, +27 to contrast, -100 to highlights, +22 to shadows, +9 to blacks, +9 to whites. The white balance "daylight" was defined to be at 5500K, and +10 to tonality (i.e. in the direction of magenta).

The image was then exported as a TIFF image with 16 bits per channel and no compression. This was done because the raw format of Canon is proprietary.

This image was then imported into MATLAB and then into "Color Thresholder". Areas that were not entirely covered by the paint were selected, and the resulting mask was inverted. This was done to remove most outliers. The portion removed was not significant,

however. 731069 points, 4.0806% were removed.

Script A.65 was then used the same way as before. This resulted in an $n \times 3$ matrix of reference colors.

Figure 3.11: A closeup of all the GCPs in the dataset "E6". From the top left corner to the right, the name of the GCPs are "20" and "21". The next row has the GCPs "19" and "18". The last row shows "22" and "17". The location of each of these points are shown in Figure 3.4. The GCP is in the middle of each "subimage".

Figure 3.12: An overview of where the different GCPs are in the “E6” dataset. This is a closeup of Figure 3.4. From the top, towards the bottom of the image, the GCPs are called “20”, “21”, “19”, “18”, “22”, and “17”.

Chapter 4

Results

A PROTOTYPE of a computer program that is able to georeference an orthophoto given the measured-in ground control points (GCPs) and a dataset of a reference color. The prototype gives a two dimensional (2D) absolute orientation unless a digital elevation model (DEM) is supplied as input. In which case it gives a three dimensional (3D) absolute orientation.

The prototype have been applied to two different orthophotos, and the results from the georeferencing are presented in this chapter. Different algorithms for calculating the absolute orientation parameters were used. Different sets of reference colors where also used.

The larger figures of this chapter are placed at the end of the chapter for readability. These figures are stitched together to form a mosaic such that they do not take up too much space in the main part of this thesis. Fine details might be difficult to see in some of these figures. Enlarged versions of the mosaics are therefore given in Appendix B.8.

4.1 The prototype

The entire code base for the prototype is given in Appendix A. The code base can also be found at `https://github.com/cLupus/AutoRef`. In order to function properly, it needs to know which color the GCPs are marked with. Multiple such datasets where created, and they are presented in Section 4.2.

The prototype is able to georeference the orthophoto "Lerkendal"

With 12 runs of the prototype with the orthophoto "Lerkendal", and the set of reference colors describved in Section 3.4.2.1 took an average of 407.2474 seconds (6.7875 minutes). This was with the four least-squares estimation (LSE) algorithms Horn, Horn-Hilden, Umeyama, and Umeyama*. The data was run with the option "Rematching" turned on, and off. The standard deviation of the running time was 48.9237 seconds . When the reference color described in Section 3.4.2.2, the average time was 769.9 ± 79.5

seconds, or slightly less than 13 minutes.

4.2 Reference color

This section describes the sample data for the Mahalanobis distance metric were produced. From (2.2) we see than the metric requires a set of reference data. In this case the set is a reference color, or a set of samples of colors. The entire sample is not necessary to define the metric, however. Only the covariance matrix, and the mean value of the data is necessary to define it. Such statistics are given in Table 4.2 and 4.3 for two different sample sets that were produced.

The two sets are too large to be given here in their entirety. Instead, summary statistics, histograms, and scatter plots are provided in Section 4.2.1. The datasets are also available as comma separated values at `http://server.nistad.me/AutoRef/`. The dataset from the orthophoto "Lerkendal" is called `sample-gcp.csv`, while the dataset from the marked GCP i called `extracted-values-from-RAW.csv`.

4.2.1 Description of sample data

Figure 4.4 and 4.5 shows the distribution of each color channel and their pairwise correspondence. The first is of the dataset produced by extracting values from the orthophoto "Lerkendal", while the latter is of the dataset produced by extracting the color from a marked GCP.

From Figure 4.4b and 4.5b, we see that the red green blue (RGB) channels are strongly correlated. The data was converted to the Lab color space in order to decouple chormaticity and brightness. This was done by using the MATLAB function `rgb2lab`.

Table 4.2 and 4.3 shows that converting the values to Lab had a significant effect on making the values less correlated. Green and blue have a correlation coefficient of 0.9657 for the first set of values. By contrast, a and b have a correlation coefficient of 0.4670. The change is most substantial between red and blue and between Lightness and b; from close to 1, to close to 0. Another reason for converting the RGB to Lab is to avoid the spike of 1's of reds in the set from the orthophoto. Figure 4.4b also suggests that the green and blue channel consists of two distributions, as there is a local maxima in the darker colors.

Visualization (mean + std and/or all the data sorted)

Normal distribution In Section 2.4.1, it was assumed that the reverence data used with the Mahalanobis distance comes from a Gaussian (normal) distribution. The Kolmogoroc-Smirnov test was therefore used to determine whether the data might come from a standard normal distribution (Massey, 1951). I.e. $\mu = 0$, $\sigma = 1$. Each channel was normalized by subtracting the mean, and divided by the standard deviation. This was done for both sets as RGB and Lab.

The Kolmogoroc-Smirnov test is implemented in MATLAB as `kstest`. As one might expect from the histograms in Figure 4.4 and 4.5, none of the channels comes from a Gaussian distribution. For the second set of values (i.e. Figure 4.5 and Table 4.3) the

p-value was a plain 0 for both RGB and Lab. The plain 0 is likely a result of the fact that double precision numbers cannot have an absolute value less than $4.9407 \cdot 10^{-324}$. For the first set, the p-values were many orders of magnitude larger; $1.5892 \cdot 10^{-90}$ for the red channel, $9.1773 \cdot 10^{-44}$ for green, and $9.1773 \cdot 10^{-44}$. For Lab, the p-values were even greater; $8.9178 \cdot 10^{-45}$ for Lightness, $2.1934 \cdot 10^{-25}$ for a^*, and $4.2608 \cdot 10^{-42}$ for b^*. In other words, they do not come from a normal distribution.

4.3 Finding thresholds, and “arbitrary” values

How was Δr and $\Delta\theta$ found (to be)? Empirically. 0.05 and 0.05 was found to work sufficiently well, and be a good compromise between ensuring a correct matching, while still being liberal enough to account for most of the imperfections... Function "Find optimal parameters.m"

Thresholds for mahal to create binimg.

4.4 Georeference Real-World cases

In the following two sections, the results from running the prototype with various input and setting are given. In both sections, the reference color described in Section 3.4.2.1 is used. The prototype was then run once for each of the algorithms for calculating the absolute orientation parameters. The algorithms were Horn, Horn-Hilden, Umeyama, and Umeyama*. For “Lerkendal”, this process was done twice; once for the option “rematching” turned off, and once on.

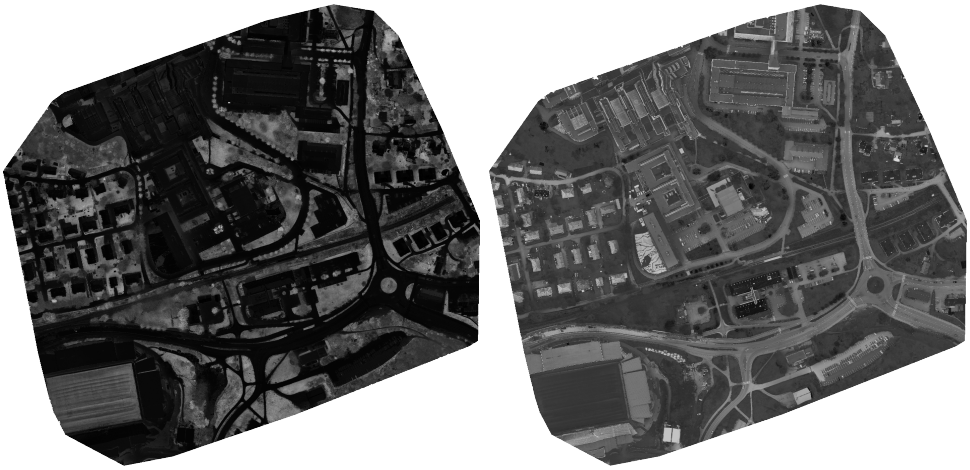
The orthophoto “Lerkendal”

4.4.1 The Lerkendal dataset

The prototype was run 8 times for the set “Lerkendal” when the sample data from the orthophoto itself is used. The three different algorithms, and Umeyama* where used with the option “Rematch” was first set to `false`, and then `true`. This option determines whether Figure 3.2, or 3.3 is to be run. By using “Rematch”, the prototype first tries to find all the GCPs as it does without the potion. After some points are found and the absolute orientation parameters are found, they are inverted. That is (2.24) is rearranged such that $[x, y, z]^\intercal$ is on the left-hand-side of the equation. The set of GCPs are then transformed to image coordinates. Then, a certain area around that point is examined more thoroughly to find where the GCP most likely is located.

Figure 4.7 and 4.6 shows the location of the GCP candidates that passed all the morphological tests, and matches the topological point pattern (TPP) of the measured-in coordinates of the GCPs. These are marked with blue x’s.

From the candidate matching (CM), the measured-in coordinates are extracted. Then the absolute orientation parameters are inverted, and the measured-in coordinates are transformed into image coordinates and plotted as orange pluses. This gives an indication of



(a) Using reference data from the orthophoto **(b)** Using the reference data from a marked GCP

Figure 4.1: The orthophoto "Lerkendal" converted into a normalized distance plot. For both orthophotos, the Mahalanobis distance metric was used. The reference sample described in Section 3.4.2.1 was used in (a), while (b) used the sample described in Section 3.4.2.2. The darker areas represents distances close to zero, while bright areas represents normalized distances close to 1.

how well the absolute orientation is. The significance of these points are explained in Section 5.1. For now, notice how the two sets of points line up, *except* for Figure 4.6c. The fact that the markings in Figure 4.7c does not match any of the markings in Figure 3.5.

For larger versions of the figures in Figure 4.7 and 4.6 see Appendix B.8.

Figure 4.8, 4.9, and 4.10 shows the direction and relative magnitude of the residuals of Figure 4.7 and 4.6. The underlying numbers for these figures are given in Table B.12 for Figure 4.9a and 4.9b. Table B.14 for Figure 4.9c and 4.9d. Table B.16 for Figure 4.9e and 4.9f. Table B.18 for Figure 4.10a and 4.10b. Table B.20 for Figure 4.8a and 4.8b. Table B.22 for Figure 4.8c and 4.8d. Table B.24 for Figure 4.8e and 4.8f. Table B.26 for Figure 4.10a and 4.10b.

Additionally, the magnitude of the GCPs with the smallest, and largest residual is given in Table B.29 for Figure 4.8, B.6, and B.8, while Table B.28 shows the same for Figure 4.9, B.5, and B.7.

4.4.1.1 Reference color

Figure 4.1 shows the normalized Mahalanobis distance of the orthophoto "Lerkendal". Figure 4.1a used the reference colors extracted from the orthophoto itself, as described in Section 3.4.2.1, while Figure 4.1b was made from sampling a marked GCP, as described in Section 3.4.2.2. From this, we see that the sample data from Section 3.4.2.1 gives a "liberal" estimate of which areas can be considered a GCP. Figure 4.1b, on the other hand, gives a stricter, or more "conservative" estimate of what areas might be a GCP.

4.4.1.2 Which where found?

Name	NEW1	NEW2	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18
Found	4	3	2	0	3	3	0	0	1	4	4	4	0	4	3	3	3	2	0	4

Table 4.1: Shows how many times the different GCPs were found in Figure 4.7.

In the four runs without rematching above, some points were consistently found, while others were consistently *not* found. The points that were found consistently are "NEW1", "P8", "P9", "P10", "P12", and "P18". On the other hand, "P2", "P5", "P6", "P11", and "P17" were consistently *not* found. Figure 4.2 shows an overview of what these points look like. Blue squares represent GCPs that were consistently found, while red squares represent those that were consistently not found.

Figure 4.2: An overview over which GCPs where consistently found (blue) and those that were consistently *not* found (red). The order of the GCPs is the same as in Figure 3.7; i.e. "New1" at the top left corner, and "P18" in the lower right corner with the rest of the points in ascending order. Figure B.17 and B.18 show larger versions of these two images.

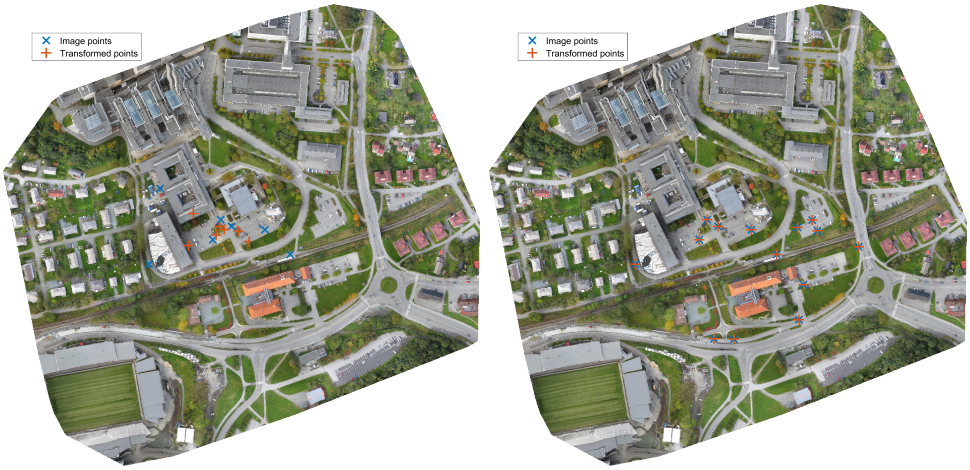
There does not seem to be a particular pattern: The least visible are P9, P17, P8, possibly also P4. 4 0 4 3 On the other hand, some of the most visible points (P5, P6, P11) are not detected at all.

4.4.1.3 Distribution of the residuals

When a model is correct, the residuals are expected to be normally distributed (Kyriakidis, 2015). Figure 4.11 shows histograms of the residuals for Northing, Easing, and Elevation. The first column have had 10% of each tail removed as outliers. No pruning have been done with the second column. 10% was chosen because it removed the small "bumps" seen to the right and left of the two primary columns in Figure 4.11b and 4.11d. All six of these where then tested to be normal with the mean value, and the standard deviation being equal to the sample mean and standard deviation by the one sample Kolmogorov-Smirnov test is MATLAB (Massey, 1951). The null-hypothesis is that the data comes from a normally distributed sample with mean equal to zeros, and standard deviation equal to 1. The null-hypothesis was *rejected* for all, *except* for the pruned values of elevation (Figure 4.11e). The significance level (p-value) was 0.0021 for pruned Northing, 0.0040 for pruned Easting, 0.4017 for runed Elevation, $6.1344 \cdot 10^{-15}$ for Northing, $1.3656 \cdot 10^{-14}$ for Easing, and $1.1279 \cdot 10^{-04}$ for Elevation.

4.4.2 The "E6" dataset

Calculation of corrections is a trade secret.



(a) Illustrates the placement of image GCPs, and transformed GCPs when the scale factor proposed by Umeyama (1991) is used.

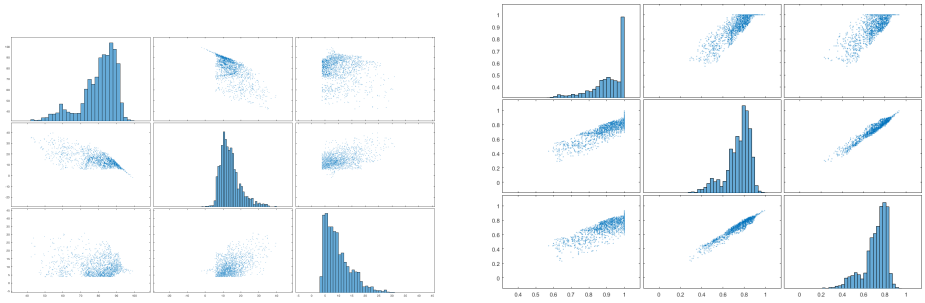
(b) Illustrates the placement of image GCPs, and transformed GCPs when the scale factor proposed by Horn (1987) is used.

Figure 4.3: An illustration of how the choice of method for calculating the scaling factor, s, affect the placement og GCPs. The scaling factor, $s = {}^{\mathrm{tr}(\boldsymbol{DS})}/_{\sigma_x^2}$, that was proposed by Umeyama (1991) is used in (a). In (b), the scaling factor, $s = \sqrt{\sigma_x^2/\sigma_y^2}$, as proposed by Horn (1987), is used. Except for the choice the scaling factor, s, everything is identical between (a), and (b). The orthophoto "Lerkendal" is used in both instances. Blue crosses signify the location of a GCP that was extracted by the prototype. Orange pluses signify where the set of corresponding measured-in GCPs are in the image when the inverse of the absolute orientation parameters is used to transform the measured-in GCPs into image coordinates.

4.4.3 Choice of sample data

4.4.4 Visibility og GCPs

E6 vs Lerkendal + "gcp.png" / "gcp-E6.png" Should be relatively "big", some 20 - 40 pixels across after the aerial images have been made into a orthophoto. The aerial images taken over Lerkendal were taken in two directions. That is, the drone took two passes, one "vertically", and one "horizontally". This makes it possible to super sample the result. Boucher et al. (2008).

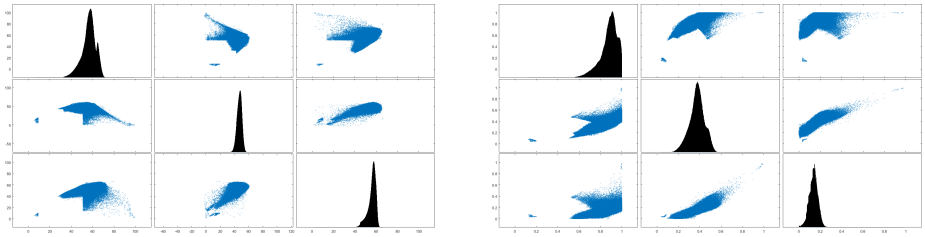


(a) Lab **(b)** RGB

Figure 4.4: Box plot of the reference sample created directly from the orthophoto "Lerkendal". The diagonal shows the histogram of the different bands, while the off-diagonal entries show the scatter plot of the different bands against each other. Both (a) and (b) display the same data, but in (a), the data have been converted to the Lab color space. (b) uses the RGB color space.

	Red	Green	Blue	Lightness	a	b
Mean	0.9080	0.7426	0.7113	80.1143	14.0786	9.8678
Minimum	0.5725	0.2902	0.2314	42.2719	-1.8771	3.8361
Maximum	1.0000	0.9922	0.9333	99.0975	39.5758	30.8549
Cov. (r/L)	0.0091	0.0099	0.0086	101.0732	-36.8650	-4.7863
Cov. (g/a)	0.0099	0.0138	0.0128	-36.8650	32.6908	13.1508
Cov. (g/b)	0.0086	0.0128	0.0128	-4.7863	13.1508	24.2553
Corr. (r/L)	1.0000	0.8835	0.8026	1.0000	-0.6413	-0.0967
Corr. (g/a)	0.8835	1.0000	0.9657	-0.6413	1.0000	0.4670
Corr. (b/b)	0.8026	0.9657	1.0000	-0.0967	0.4670	1.0000
Number of samples				3710		

Table 4.2: A table showing descriptive statistics for the sample values from the orthophoto. Correlation coefficients between the different bands are also included. To the left of the dashed line are the statistics for the RGB values of the sample. To the right are statistics of the same sample set when the values were converted into Lab. Cov. stands for covariance, and Corr. for correlation coefficient. The parentheses indicates which channel the covariance, and correlation coefficients are relative to. To the left of the dashed line, they are relative to red, green, and blue, while to the left, it is Lightness, a^*, and b^*.

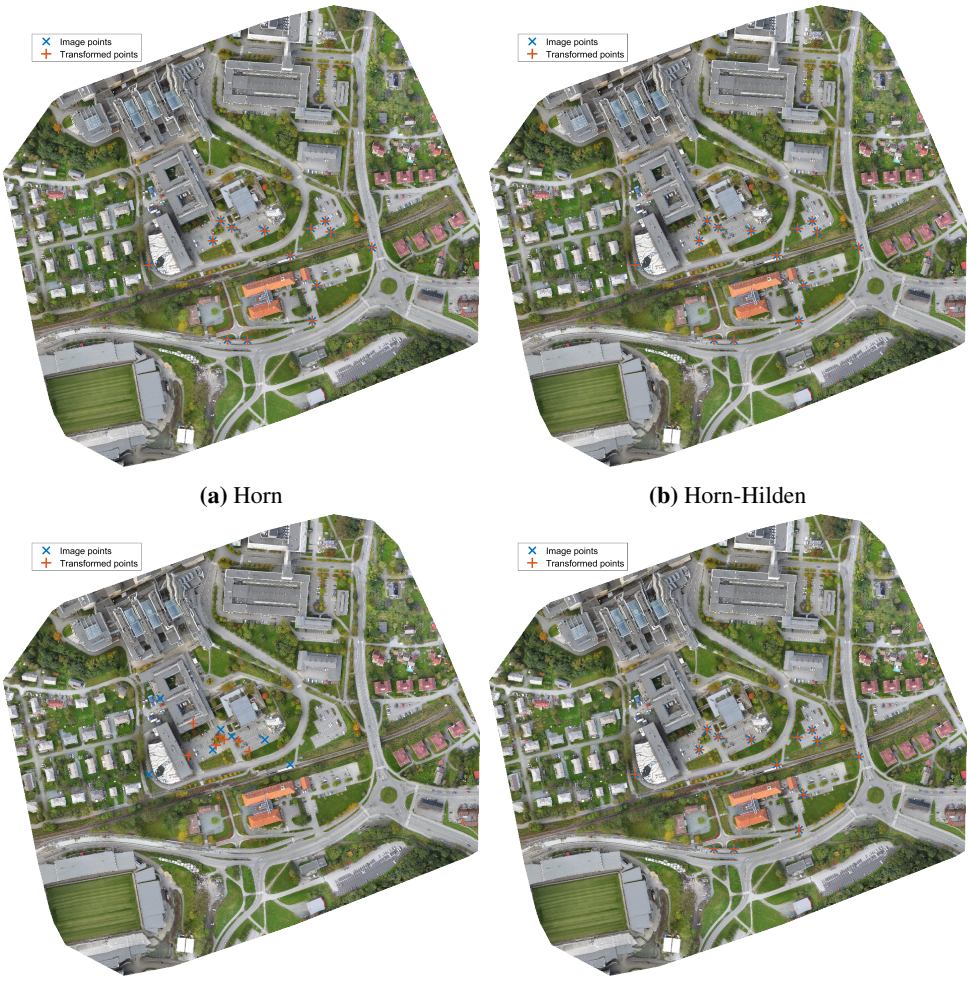


(a) Lab **(b)** RGB

Figure 4.5: Box plot of the reference sample created from the paint of a marked GCP. The diagonal shows the histogram of the different bands, while the off-diagonal entries show the scatter plot of the different bands against each other. Both (a) and (b) display the same data, but in (a), the data have been converted to the Lab color space. (b) uses the RGB color space.

	Red	Green	Blue	Lightness	a	b
Mean	0.7047	0.7010	0.7981	73.8650	5.4686	-12.0311
Minimum	0	0.4743	0.4847	47.4435	-35.9723	-24.8647
Maximum	0.9608	0.7569	0.6824	80.5595	48.4793	30.2006
Cov. (r/L)	0.0092	-0.0024	0.0021	4.7691	-8.4675	3.0198
Cov. (g/a)	-0.0024	0.0019	-0.0001	-8.4675	169.8797	-41.4681
Cov. (g/b)	0.0021	-0.0001	0.0128	3.0198	-41.4681	12.8809
Corr. (r/L)	1.0000	0.5777	0.7880	1.0000	-0.2975	0.3853
Corr. (g/a)	-0.5777	1.0000	-0.1138	-0.2975	1.0000	-0.8865
Corr. (b/b)	0.7880	-0.1138	1.0000	0.3853	-0.8865	1.0000
Number of samples	17 184 835					

Table 4.3: A table showing descriptive statistics for the sample values from the image of a marked GCP. Correlation coefficients between the different bands are also included. To the left of the dashed line are the statistics for the RGB values of the sample. To the right are statistics of the same sample set when the values were converted into Lab. Cov. stands for covariance, and Corr. for correlation coefficient. The parentheses indicates which channel the covariance, and correlation coefficients are relative to. To the left of the dashed line, they are relative to red, green, and blue, while to the left, it is Lightness, a^*, and b^*.



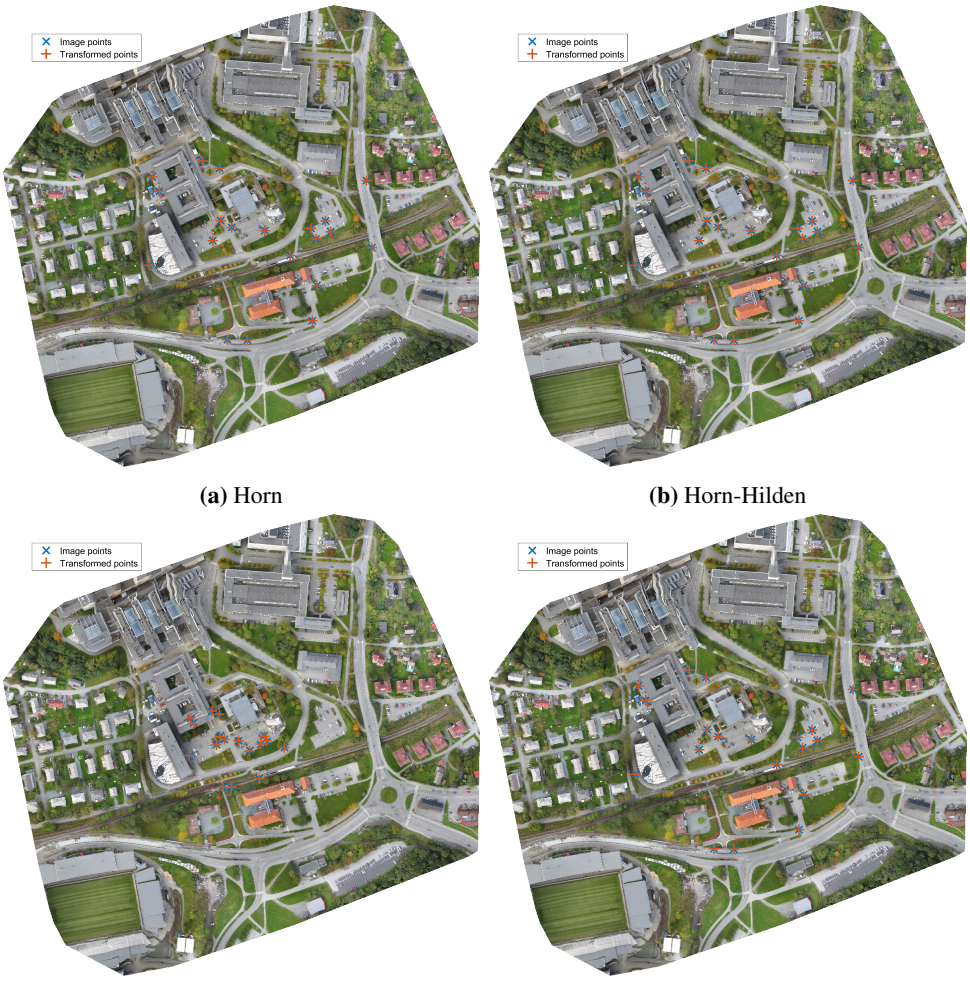
(a) Horn

(b) Horn-Hilden

(c) Umeyama

(d) Umeyama*

Figure 4.6: Shows the resulting placement of the GCPs in the orthophoto "Lerkendal". Rematching was turned *off* for all these. In (a) Horn was used to find the absolute orientation parameters. Horn-Hilden was used in (b), Umeyama in (c), and (d) used Umeyama*. For enlarged versions see Figure B.10, B.11, B.13, and B.15.



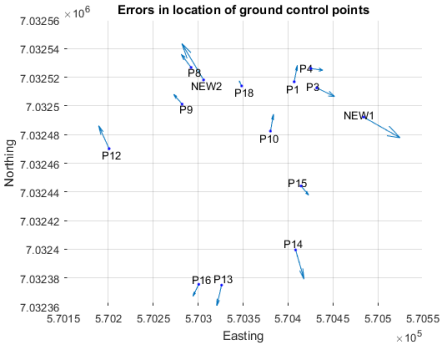
(a) Horn

(b) Horn-Hilden

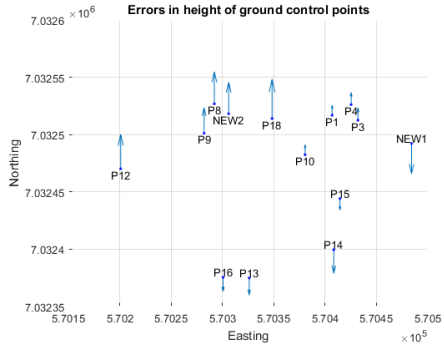
(c) Umeyama

(d) Umeyama*

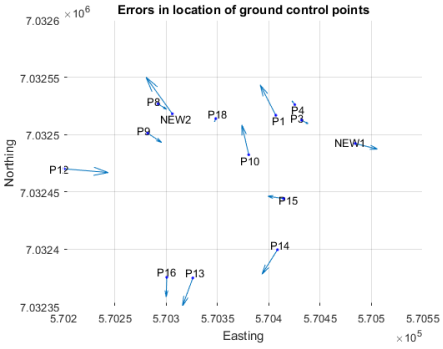
Figure 4.7: Shows the resulting placement of the GCPs in the orthophoto "Lerkendal". Rematching was turned *on* for all these. In (a) Horn was used to find the absolute orientation parameters. Horn-Hilden was used in (b), Umeyama in (c), and (d) used Umeyama*. For enlarged versions see Figure B.9, B.12, B.14, and B.16.



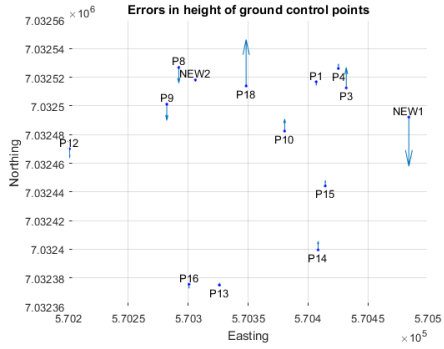
(a) Horn, location



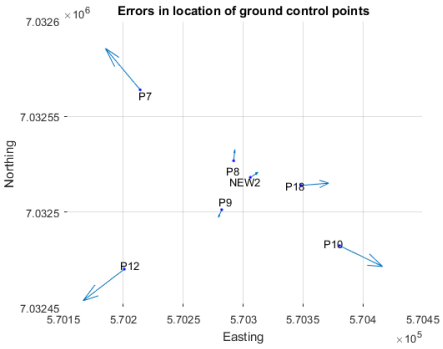
(b) Horn, elevation



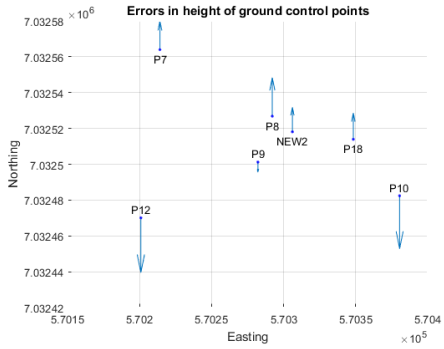
(c) Horn-Hilden, location



(d) Horn-Hilden, elevation

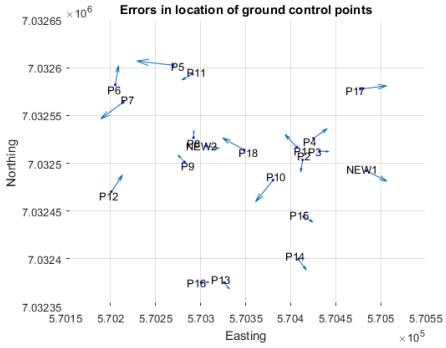


(e) Umeyama, location

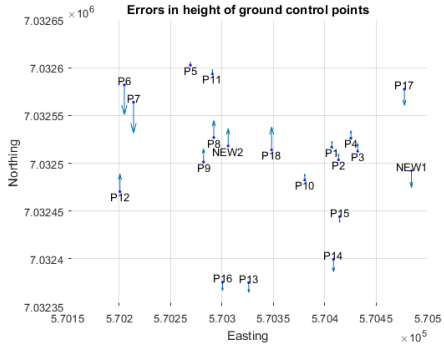


(f) Umeyama, elevation

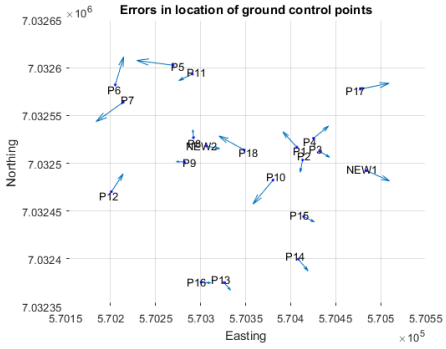
Figure 4.8: Plots of residual errors for location (Northing - Easting) and Ellipsoidal Height. (a) and (b) shows the residuals by using Horn to calculate the absolute orientation parameters. (c), and (d) shows the same for Horn-Hilden, while (e) and (f) used Umeyama. None of them used rematching. For a larger version of these, see Figure B.1, B.3, and B.5.



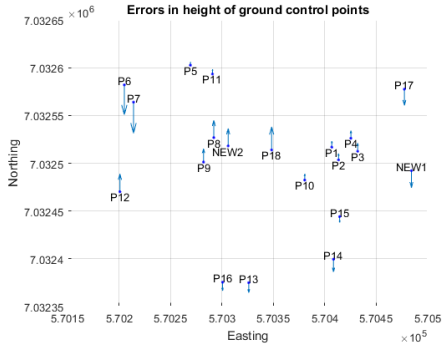
(a) Horn, location



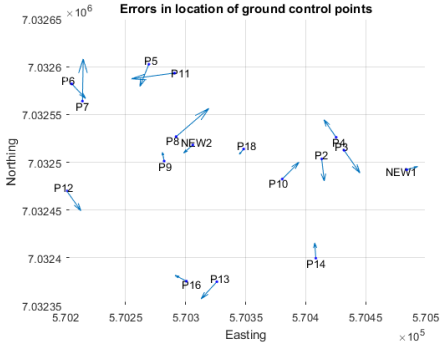
(b) Horn, elevation



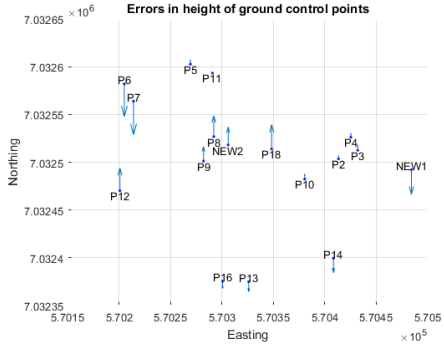
(c) Horn-Hilden, location



(d) Horn-Hilden, elevation

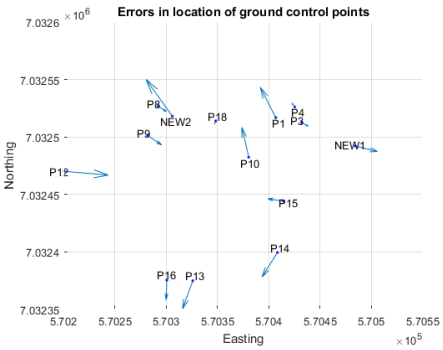


(e) Umeyama, location

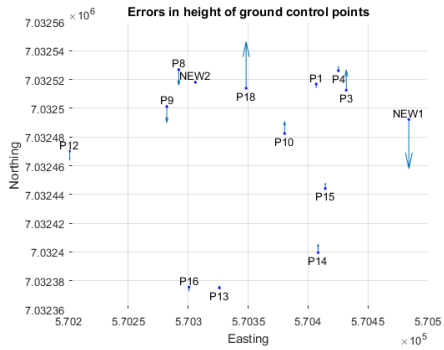


(f) Umeyama, elevation

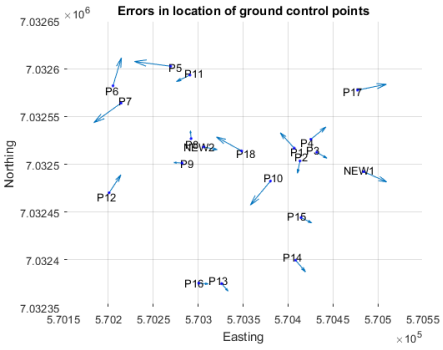
Figure 4.9: Plots of residual errors for location (Northing - Easting) and Ellipsoidal Height. (a) and (b) shows the residuals by using Horn to calculate the absolute orientation parameters. (c), and (d) shows the same for Horn-Hilden, while (e) and (f) used Umeyama. All of them used rematching. For a larger version of these, see Figure B.2, B.4, and B.6.



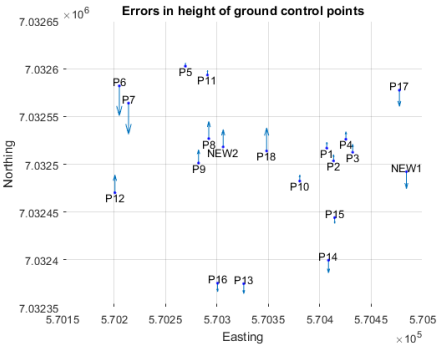
(a) Umeyama*, location



(b) Umeyama*, elevation

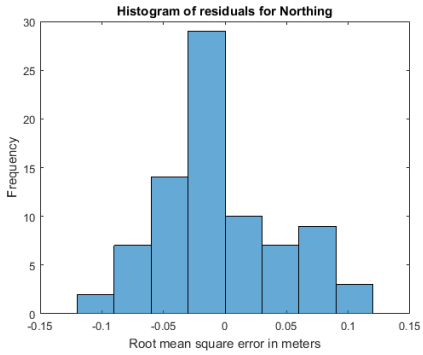


(c) Umeyama* rematched, location

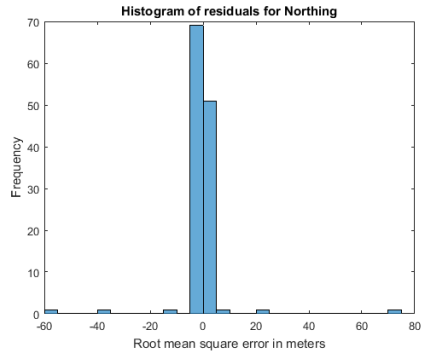


(d) Umeyama* rematched, elevation

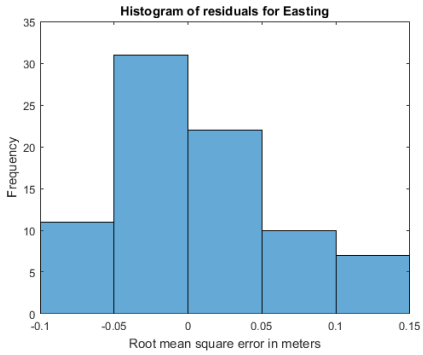
Figure 4.10: Plots of residual errors for location (Northing - Easting) and Ellipsoidal Height. (a) - (d) shows the residuals by using Umeyama* to calculate the absolute orientation parameters. (a) and (b) did not use rematching, while (c) - (d) did. For a larger version of these, see Figure B.7 and B.8.



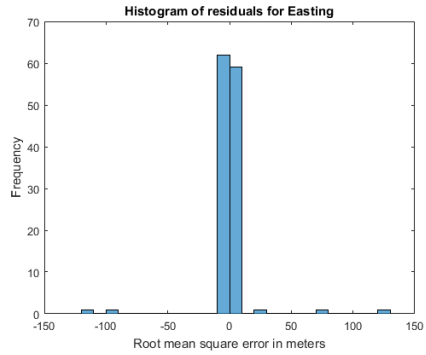
(a) Northing 10%



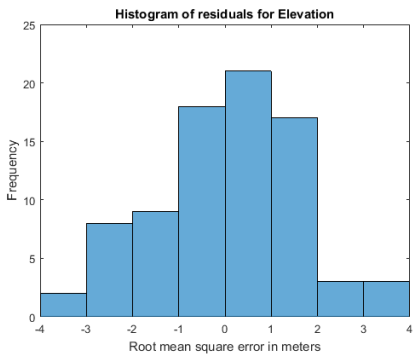
(b) Northing, full



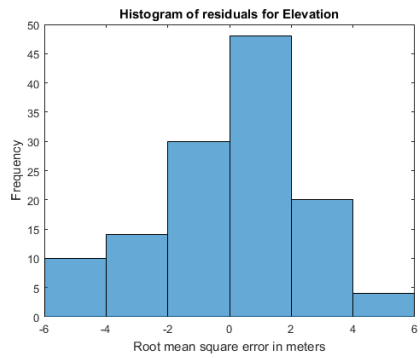
(c) Easting 10%



(d) Easting, full



(e) Elevation 10%



(f) Elevation, full

Figure 4.11: Shows the histograms of the residual errors for the Northing, Easting, and Elevation component of the data from Figure 4.8 - 4.10. (a), (c), and (e) shows the histogram of the residuals when 10% of the ends have been cut off. (b), (d), and (f) shows the same histogram without any pruning.

(a) Horn

(b) Horn-Hilden

Figure 4.12: Shows the resulting placement of the GCPs in the orthophoto “E6”. Rematching was turned on. (a) shows the results when using Horn, while (b) is obtained by using Horn-Hilden.

(a) Umeyama

(b) Horn with limited GCPs

Figure 4.13: Shows the resulting placement of the GCPs in the orthophoto “E6”. Rematching was turned on. (a) shows the results when using Umeyama, while (b) is obtained by using Horn, but the list og GCPs was limited to only the points visible in the orthophoto.

Chapter 5

Discussion & Analysis

IN SHORT, the prototype does work. It is capable of finding where the ground control points (GCPs) are in an orthophoto. The prototype has also been shown to be feasible; able to georeference an orthophoto in less than 7 minutes. However, the prototype is far from perfect.

In this chapter, the results are analyzed and discussed. In particular, what the different results means in practice will be discussed. Possible reason for some of the discrepancies are presented, along with possible solutions.

5.1 Analysis of placement

Figure 4.7 and 4.6, along with their enlarged couterparts in Figure B.10 - B.16 show that the choice of algorithm for absolute orientation does not play a major part.

In the following sections, the results from Section 4.4 are discussed and analyzed. The results from using the orthophoto "Lerkendal" and the sample data from Section 3.4.2.1 is given first. Then the results from applying the prototype to the orthophoto "E6" with the same sample data are discussed.

5.1.1 Horn and Horn-Hilden

In particular, Horn, and Horn-Hilden gives very similar results. This is evident when comparing Figure 4.7a against 4.7b, or their enlarged counterparts. When the GCPs are rematched, the residual plots in Figure 4.9a - 4.9d are nearly identical.

Their "un-rematched" counterparts in Figure 4.8a - 4.8d, however, are quite dissimilar. The direction, and magnitude of the residual errors varies quite a bit; "P9" reverses direction, and "P1" has one of the largest magnitudes of Horn-Hilden (Figure 4.8c), while in Horn (Figure 4.8a) it is one of the smallest. The reverse is true for "NEW1". The residual

of elevation is generally much smaller by using Horn-Hilden (Figure 4.8d) than Horn (Figure 4.8b).

Horn and Horn-Hilden found the same GCPs. This is to be expected, as points are found in the matching stage of the prototype. This stage is independent of the choice of absolute orientation algorithm. The algorithm simply chooses the similarity transform (ST) which gives the smallest total root mean square error (RMSE). This indicate that both algorithms agree on what constitute the best absolute orientation parameters. These parameters are also quite similar, as can be seen by comparing (B.14), (B.15), and (B.16) against (B.17), (B.18), and (B.19).

The fact that both algorithms found all the GCPs when rematching was turned on suggests that the initial absolute orientation parameters were close to their optimal values. This is also evident from the RMSE of the location (Northing-Easting) calculated in Table B.21 and B.23. It is 10.11 cm, and 10.13 cm respectively.

5.1.1.1 Comparing the absolute orientation parameters

Since the scale factor s is calculated the same way in Horn (1987) and Horn et al. (1988), one would expect them to have the same value. The only way it would be different is if different sets of candidate matchings (CMs) minimized the total RMSE. Section 2.6.1.2 and 2.6.1.3 shows that they use the same equation for calculating the translation vector $\boldsymbol{t}$ as well as the scale factor. Thus, it is only for them to give different results if they obtain different rotational matrices, $\boldsymbol{R}$. If the RMSE is different for the two algorithms, both cannot give optimal absolute orientation parameters, which both claim they do.

When rematching is turned on, i.e. all the marked GCPs are found and the absolute orientation parameters are calculated, the difference in total RMSE is a minuscule 0.005993%. The RMSE in location is much greater, but not significant (0.195074%). When rematching is turned off, however, differences becomes apparent. The total RMSE shows a 68.358998% difference in favor of Horn-Hilden, while the difference in error of location is 41.450787%. Also in favor of Horn-Hilden. This suggests that the two algorithms favors different sets of point, or that there is glitches, or bugs in Script A.13 or A.14. Another explanation for he difference in outputs is numerical accuracy in calculating the rotation matrix.

5.1.2 Umeyama

In short, Umeyama preformed worse than both Horn and Horn-Hilden. The main reason for this seem to be due to the scaling factor proposed in Umeyama (1991). The calculated scale factor is more than twice the size of that obtained by using Horn or Horn-Hilden (0.0751 vs. 0.0320)

In his article, Umeyama writes

> Finally, since $\epsilon^2(s)$ is a quadratic form of s, the minimum value of $\epsilon^2(s)$ is *obviously* achieved when $s = \frac{\mathrm{tr}(\boldsymbol{DS})}{\sigma_x^2}$

In the article c is used instead of s for the scale factor, and the emphasis has been added.

$\epsilon^2(s)$ refers to the RMSE caused by the scaling factor. The algorithm is implemented directly in Script A.59.

Comparing Figure 4.7c, Figure 4.6c, Table B.17, and Table B.25 against Figure 4.7d, Figure 4.6d, Table B.19, and Table B.27, it becomes plain that his claim is not true. By using the same scale factor as defined in Horn (1987) and Horn et al. (1988) the total RMSE can be *reduced* from 3.49 meters to 2.75 meters. In the same case, the RMSE for the location has a more drastic reduction; from 2.15 meters to 0.101 meters. That is a reduction of 95.3%! This was the case when comparing Figure 4.8e and 4.8f against Figure 4.10a and 4.10b. The RMSEs are given in Table B.25 and B.27 respectively.

The difference between Umeyama and Umeyama* is even grater when the rematching option is turned on. The RMSE of location is then reduced from 89.2 meters to 0.0738 meters. In other words, a 99.917% reduction.

By using Umeyama* instead of Umeyama, the algorithm gives results that looks suspiciously similar to that of Horn-Hilden. This is evident when comparing the plot of residual errors in Figure 4.8c with Figure 4.10a, Figure 4.9c with Figure 4.10c, Figure 4.8d with Figure 4.10b, and Figure 4.9d with Figure 4.10d.

In fact, the residuals in Table B.14 and Table B.18 are identical. This is a consequence of the absolute orientation parameters are identical. This can be seen by comparing (B.4) with (B.11), (B.5) with (B.12), and (B.6) with (B.13). Even when rematching is turned off, the algorithms gives identical results.

5.1.2.1 Why not an iterative algorithm?

One of the primary reasons why the absolute orientation algorithm given in Kraus (2007) was not implemented, and compared to the other algorithms is that it uses an iterative approach. Thus, it needs an initial guess to start the iteration. Initial guesses can be found automatically, however. One approach is to select some arbitrary values initially. Kraus suggests a more robust method for finding an initial guess. The method requires at least four points. Select excatly four of these. The choice cannot be made arbitrary as a poor choice can cause the iterations to not approach a particular set of values for the absolute orientation parameters.

Another concern with the iterative approach, is the use of linearizion which may cause the algorithm to return a sub-optimal solution. This happens if the iteration getting "stuck" in a local minima in the parameter space. Thus the absolute orientation parameters cannot be guaranteed to be optimal for any set of GCPs candidates found in the image and measured-in GCPs.

The main advantage of using an iterative approach, such as the algorithm proposed in Kraus (2007) is that it is easy to calculate, and implement. Unlike Horn, Horn-Hilden, and Umeyama, eigenvalues and eigenvectors are not necessary for the algorithm in Kraus (2007). If done by hand, or if there are no functions or libraries available for computing eigenvalues and eigenvectors, this is a major advantage. However, since the matrices are small (3×3 and 4×4), and there are ready made programming libraries that calculates eigenvalues and eigenvector, this is not a major advantage.

The majority of the running time of the prototype is dedicated to finding GCP candidates

(7, or more minutes), while matching the candidates to the measured-in GCPs *and* finding the absolute orientation parameters is done in approximately 2 seconds. Thus, simplicity and computational efficiency for the algorithm that finds the optimal absolute orientation parameters is not important towards how feasible, or efficient the prototype is. A consistent and accurate method for matching the two sets of points, and finding optimal absolute orientation parameters is of greater importance to the feasibility of the prototype, and future production-ready systems.

5.1.3 E6

Figure 4.13 and 4.12 shows where the prototype placed the different GCPs, and where the measured-in GCPs are in the orthophoto by using the calculated absolute orientation parameters. Comparing the placement of the blue x's with the placement in Figure 3.4 and 3.12, and considering that some GCPs are placed *outside* the limits of the orthophoto, it becomes obvious that the prototype does not work perfectly. The reason for these results seem to stem from the prototype's failed attempt at locating where the GCPs are in the orthophoto. The reverence values used in the generation of these results is the same as was used with the results from "Lerkendal".

Figure 4.1a and B.17 shows the Mahalanobis distance metric used applied to the orthophoto "Lerkendal" with the same reference color. From this, one can see that the reference color favors concrete and worn asphalt as well as roofs.

From Section 3.4.1, we know that the signalized area around the GCPs in "E6" is smaller than those of "Lerkendal" (≈ 10 pixels across versus ≈ 20 pixels). Visually, this is even clearer; Figure 3.11 and 3.7 shows a closeup mosaic of the GCPs of the orthophotos "E6" and "Lerkendal" respectively. The parts of the two mosaics have the same image resolution; 201×201 pixels. Even though Figure 3.11 has been enlarged compared to Figure 3.7, the markings in Figure 3.11 are not as easy to spot as the markings in Figure 3.7.

Combined, these two aspects give a plausible explanation for the prototype not being fully able to find the GCPs in "E6". Another reason might be that the prototype uses fixed, hard thresholds for converting a distance-image into a binary image. This binary image is then put through different morphological tests.

A possible solution to this problem is to use soft thresholds, and adaptive limits for creating binary images. Another option is to carefully design the reference color and the distance metric to better describe what constitute a GCP.

5.2 Analysis of residuals

In this section, the residuals from "Lerkendal" are analyzed. The residuals from "E6" were not included because, as seen in Figure 4.12 and 4.13, the prototype was unable to extract the marked GCPs. Thus, the residuals are major outliers. The largest magnitude for "E6" occurred when using Umeyama. It is 2020.99 meters, and occurs at the point "22", which is a GCP that is visible in the orthophoto "E6". Interestingly, the smallest error also occurred when using Umeyama; 167,19 meters, supposedly at point "13", which is

not one of the GCPs in "E6". The rest of the residuals tend to lie between 100 and 1000 meters in magnitude.

The reason the residuals from using the reference color described in Section 3.4.2.2 is similar; the prorotype was unable to locate the marked GCPs.

5.2.1 Analysis of the magnitudes of the residuals

On first viewing, the residuals of Figure 4.8, 4.9, and 4.10, along with their enlarged counter parts in Figure B.2 - B.8, does not seem to follow a particular pattern. However, the results from Umeyama are all outliers, having a RMSE far grater than any other of the approaches.

Section 4.4.1.3 showed that the residuals are mostly not normally distributed. Consequently, some aspects of the model used to describe a GCP is not close enough to reality (Kyriakidis, 2015). The model is the use of reference color for the Mahalanobis metric, the Mahalanobis distance metric itself, and any morphological tests applied to the GCPs candidates.

When the residuals are pruned of clear outliers (Figure 4.11a, 4.11c, and 4.11e) the distributes are not very far from being normally distributed.

5.2.2 Analysis of the direction of the residuals

In this section, the direction of the residuals is analyzed. The goal is to find any trends that might suggest a bias, and thus how the prototype can be improved.

5.2.2.1 The direction of residuals

Based on Figure 4.8, 4.9, and 4.10, and their enlarged counterparts, there does not seem to be a clear and obvious trend in the direction of the residuals. Figure 5.1 shows that the angle of the residual vectors of the location in Figure 4.8, 4.9, and 4.10 generally comes from a uniform distribution. The orientation of these angles is counter clockwise from the x-axis (Easting). There is a bias towards southeast (the peak between -1 and 0). Since the x-axis of the histogram is cyclic (radians from a specific axis), the small sample size is likely responsible for the slight elevation towards the left-hand side of the figure and the small dip towards the right-hand side. Some of the values close to π could have been slightly larger, or smaller, and then received a negative value. One explanation for this bias is that the rotational matrix $\boldsymbol{R}$ have a slight clockwise bias.

Figure 5.1: Histogram of the angle (in radians) of the residuals of location in Figure 4.8, 4.9, and 4.10.

5.2.2.2 Distance from center

Figure 5.2 shows that there is no apparent correlation between the distance from the center of the orthophoto and the magnitude of the residual. For Figure 5.2a the correlation coefficient was slightly positive (0.2554), while for Figure 5.2b, 5.2c, and 5.2d, the correlation was insignificant (-0.0137, 0.1068, and 0.0017 respectively).

One explanation for the slight correlation between distance from the center and the magnitude of the residual is that the scale factor is a little bit too great. Another explanation is that when aerial images are made into a orthophoto, the edges of the orthophoto tend to be less accurate than the center due to more overlap between aerial images. However, the GCPs are mostly located toward the the center in the orthophoto "Lerkendal".

5.2.2.3 Direction outward

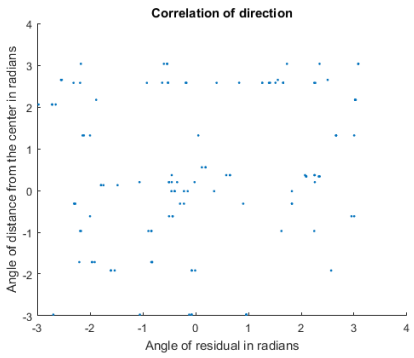
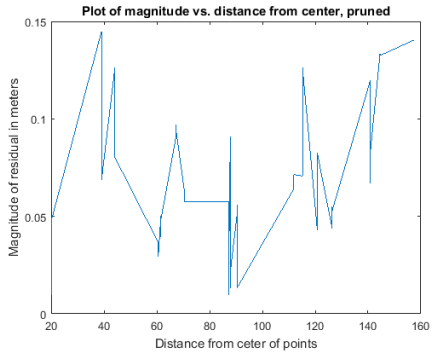
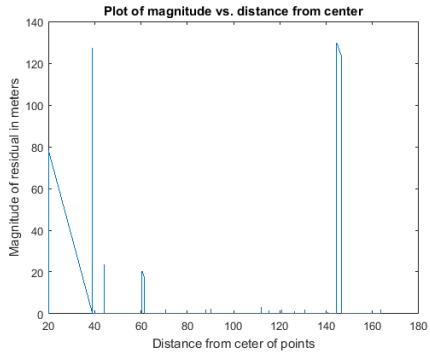


Figure 5.3: A scatter-plot of the angle of the (2D) residuals against the angle of the (2D) distance of the same point from the center.

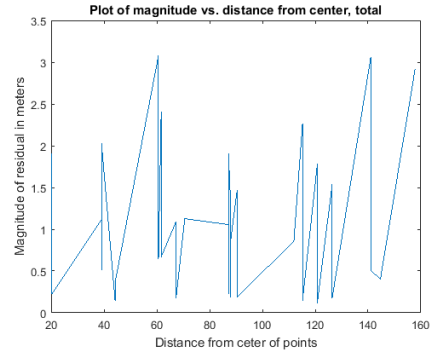
Now we examine whether there is a connection between the placement of the GCP relative to the center of the GCPs and the direction of the residual. In other words, how strongly does the residuals outward? The trend is clearly visible in Figure 4.8e. Figure 4.8a and 4.8c also show tendencies for this. However, as can be seen in in Figure 5.3, there is no strong correlation between them. The correlation coefficient between the angle of the (2D) residual, and angle of the vector formed by subtracting the center from the different GCPs' location is 0.1570.



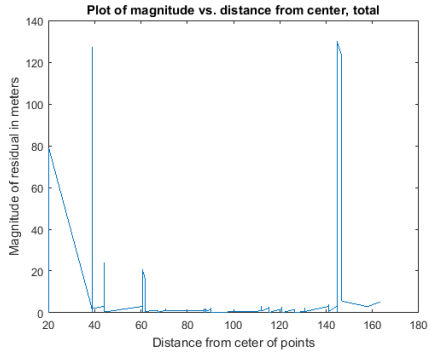
(a) Magnitude of residual location vs. two dimensional (2D) distance from center, 10%.



(b) Magnitude of residual location vs. 2D distance from center, full.



(c) Magnitude of residual vs. three dimensional (3D) distance from center, 10%.



(d) Magnitude of residual vs. 3D distance from center, full.

Figure 5.2: Plots of the relation between the magnitude of residuals and the distance from the center of the GCPs. The smallest, and greatest 10% quantiles have been removed (a) and (c). No data have been removed in (b) and (d). (a) and (b) show relationship between the magnitude of the residual in location (Northing-Easting) and 2D distance from the center of the GCPs. In (c) and (d) the Elevation component have been included in the magnitude and the distance.

5.3 On the use of reference colors

The results from using the reference color described in Section 3.4.2.2 are not included in this thesis. The reason for this is that it did not work as expected; none of the GCPs were found in "Lerkendal", nor in "E6". In some runs, the prototype was unable to match more than three of the candidates to the measured-in GCPs. By using topological point pattern (TPP), a three point matching is almost arbitrary; the first two points are, by definition, a match. The first coordinate always have the polar coordinate $(r, \theta) = (0, 0)$, while thee second always is $(r, \theta) = (1, 0)$.

For this, and other reasons, a matching proposed by TPP was not considered proper unless it had 5 - 7, or more points in it. Figure 4.1a and 4.1b shows that the reference color defined in Section 3.4.2.2 is more strict than the reference color defined in Section 3.4.2.1. That is, more areas in Figure 4.1a are considered to be close to a GCP. Therefore, the threshold values have to be stricter, or closer to zero in order to seperate potential GCPs from areas that are not. Since the hard coded thresholds for what constitutes a probable GCP are based on the fist set of reference color, it is likely too strict for the distances obtained from the Mahalanobis metric using the reference colors defined in Section 3.4.2.2.

5.4 On loading the entire orthophoto into memory

In the beginning of this thesis, the prototype was limited to loading the entire orthophoto into memory. The reason for this is that makes it easier to process the orthophoto, and keep track of where different GCP candidates are located. In theory, there is no limitation that a similar prototype, or system must load the entire orthophoto into memory (RAM).

The entire orthophoto "E6" is approximately 21 GB. In this case, and likely many others in the near future, processing only a portion of the orthophoto at a time would be beneficial. That way, the system need not run on a computer with enormous amounts of RAM (When the "E6" was converted to double, it took about 91 GB of RAM).

Programs such as PhotoScan and Pix4D are capable of exporting the orthophoto as a mosaic with an eXtended Markup Language file that shows where the different tiles are in relation to each other. By using a mosaic instead, the process becomes naturally parallel; the tiles can be processed mostly independent of each other. However, there are some edge cases that must be dealt with: What if a GCP is along the edge of a tile, and thus in two different tiles?

5.5 On marking GCPs

From the results in Section 4.4.2, Section 4.4.1, Figure 3.7, and Figure 3.11 one can infer that the marked area should be at least 20 pixels wide in the final orthophoto. The images taken over Lerkendal consisted of two passes. In these images, the signalized area was approximately 5 pixels across (3cm/pixel).

Figure 4.2 shows which GCPs were found consistently, and those that were consistently *not* found. One possible explanation for the most visible GCPs not being found is that the

reference color is a compromise between all the GCPs, and thus by using the Mahalnobis distance metric, the GCPs that have a color close to the average value is more likely to be found, instead of the GCPs with the strongest color. One possible solution to this is to paint, or lay down a black border around the different GCPs. The exact size of it needs more research to be determined, but one suggestion would be about the same size as the side length of the signalized area around the GCPs.

A round shape for the signalized area might be more appropriate than squares. The reason for this is that, as can be seen in Figure 3.7 and 3.11, the signalized areas tend to look like circles when viewed from some distance. With circles, the points could more easily be matched to a template, or small binary image. Using template can also be used with squares, but then the templates might have to be rotated to ensure detection, with will take more processing power.

5.6 Issues with TPP

One of the goals of this thesis is to investigate the applicability of the algorithm of TPP presented in Li and Briggs (2006). Additionally, no source code was supplied in the article. Nor was there any links to the working program Li and Briggs made, or the data used. Thus verification of their results can be difficult.

Even though TPP was designed around 2D georeferencing, it works well with 3D data as well. The fact that the TPP algorithm was not extend to 3D might be one of the reasons for why the elevation was generally poorer modeled than that of Northing and Easting.

During implementation, and use of the algorithm, some flaws in the original design were discovered. The first issue lies in how the point patters are sorted. The sorting and the issue is described in Section 2.5. In short, the angles are sorted by radial distance counter clockwise from a given axis. This causes a problem is a point lies just below the axis, and have an absolute radial distance which is smaller than the point before it. In this case, the point that is further away is chosen, in contrary to what is desired; the closest match.

Another issue was that, although Li and Briggs claims that a single, arbitrary point can be selected as the anchor point from the set points from the image. However, depending on the point that was chosen, the set of CMs than minimized the RMSE varied in size and accuracy. The solution to this problem was to calculate a TPP for each point in the set of GCP candidates and matching each of them against all of the TPP made from the measured-in GCPs.

5.7 Issues with the scale factor for Umeyama

In his article, Umeyama, claims that the scale factor (2.38) *obviously* minimizes the root mean square error (RMSE). No proof is given. Figure 4.8, 4.9, and 4.10 show that other least-squares estimation (LSE) algorithms that does a better job. Alternatively, the implementation developed in this thesis is flawed. The article Umeyama (1991) was used extensively during the development, and the implementation was written as directly into code as possible.

Script 5.1 gives an absolute orientation with RMSE of 89.2561. The script was run with `location` being a list of candidate GCP extracted from the orthophoto "Lerkendal". These points are given in Table B.3. The parameter `gcp` the measured-in GCPs given in Table B.1. When `'UseHornScaling', false` was changed to `true`, the RMSE of the absolute orientation became 0.8474.

The absolute orientation parameters from using the scale factor (2.31) from Horn (1987) and Horn et al. (1988) is given by (B.23) - (B.25). When the scale factor proposed in the original article, the absolute orientation parameters are given by (B.20) - (B.22). The scale factor proposed in Umeyama (1991), (2.38) is more than twice that of the scale factor obtained when using (2.31) (0.0746 against 0.0320).

Script 5.1: An exceprt from Script A.1, where a set of CMs are matched with the measured-in GCPs from Lerkendal. See Table B.1 for the coordinates.

```
[CM, ST, RMSE] = match_gcps(location, gcp, ...
'GetOptimal', true, ...
'ImageTPPMode', 'all', ...
'MinimumMatches', 7, ...
'OrientationAlgorithm', 'ShinjiUmeyama', ...
'AngleThreshold', 0.05, ...
'RadiusThreshold' , 0.05, ...
'UseHornScaling', false);
```

Chapter 6

Conclusion

THE PROTOTYPE presented in this thesis shows that an orthophoto can be georeferenced automatically based solely on the orthophoto itself, the measured-in coordinates of the ground control points (GCPs), and a set of reference color values that depict the color used to signalize, or mark the different GCPs.

Not only is it possible to georeference a orthophoto automatically, but it is feasible; a medium sized orthophoto (20 051 x 22 039 pixels) was georeferenced in less than 7 minutes with a root mean square error (RMSE) of as little as 0.0738 meters. The prototype is capable of georeference an orthophoto by itself to give a two dimensional (2D) absolute orientation and to give a three dimensional (3D) absolute orientation if a digital elevation model (DEM) is supplied.

The accuracy in elevation is not as accurate as the Northing-Easting. In the best case in this thesis the smallest total RMSE was 0.847 meters.

This thesis have also showed that the choice of algorithm to calculate the optimal absolute orientation parameters in a least-squares estimation (LSE) fashion is not arbitrary.

6.1 Further

6.1.1 Thoughts

6.1.2 Work

New implementation in C/C++ and/or Python. Thus a completely open source project.

6.2 Recommendation for marking GCPs

Bibliography

Abramowitz, M. (1972). Handbook of Mathematical Functions With Formulas, Graphs, and Mathermatical Tables. pages 17–18. National Bureau of Standards, 10 edition. Available from: http://people.math.sfu.ca/~cbm/aands/abramowitz_and_stegun.pdf.

Adobe Systems Inc. (2005). Adobe RGB (1998) Color Image Encoding. Technical report, Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704. Available from: https://www.adobe.com/digitalimag/pdfs/AdobeRGB1998.pdf.

AgiSoft LLC (2012). *Agisoft PhotoScan User Manual: Professional Edition*. AgiSoft LLC.

Allen, J. Spectral signatures of earth features [online]. *Science Mission Directorate*. (2010) [cited 2016-05-24]. Available from: http://missionscience.nasa.gov/ems/09_visiblelight.html. Image.

AntonAL. What is the best javascript image processing library? [closed] [online]. *StackOverflow*. (2015) [cited 2016-05-20]. Available from: https://stackoverflow.com/questions/3351122/what-is-the-best-javascript-image-processing-library.

Arun, K. S., Huang, T. S., and Blostein, S. D. (1987). Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-9(5):698–700.

Blackwell, C. Color vision 3: Color map [online]. (2013) [cited 2016-05-24]. Available from: https://www.youtube.com/watch?v=KDiTxWcD3ZE.

Boucher, A., Kyrakidis, P. C., and Cronkite-Ratcliff, C. (2008). Geostatistical solutions for super-resolution land cover mapping. *IEEE Transactions on Geoscience and Remote Sensing*, 46(1).

Cairns, G. (2013). Digital Photo Professional: Canon's image processing software. Available from: http://cpn.canon-europe.com/content/product/canon_software/inside_digital_photo_professional.do.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*, pages 1078–1100. The MIT Press, 3 edition.

Cruse, P. [online]. (2015) [cited 2016-04-27]. Available from: http://www.colourphil.co.uk/lab_lch_colour_space.shtml.

DroneDeploy. Getting started [online]. (2015) [cited 2015-12-16]. Available from: http://support.dronedeploy.com/docs/frequently-asked-questions.

Environmental Systems Research Institute. Redlands, C. (1995). *ARC/INFO Version 7.0.4 - Data Conversion and Regions*. GIS by ESRI. Environmental Systems Research Inst. Available from: https://books.google.no/books?id=6snBPgAACAAJ.

Erdas (2008). *IMAGINE AutoSync™User's Guide*. ERDAS, Inc., Manager, Technical Documentation. ERDAS, Inc. 5051 Peachtree Corners Circle Suite 100 Norcross, GA 30092-2500 USA.

Esri. Understanding world files [online]. (2009) [cited 2016-05-19]. Available from: http://webhelp.esri.com/arcims/9.3/General/topics/author_world_files.htm#aboutAnchor.

Esri. Faq: What is the format of the world file used for georeferencing images? [online]. (2016) [cited 2016-05-31]. Available from: http://support.esri.com/technical-article/000002860.

Esri. Georeferencing a raster automatically [online]. *ArcGIS for Desktop 10.3*. (2016) [cited 2016-04-25]. Available from: http://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/georeferencing-a-raster-automatically.htm.

European Global Navigation Satelite Systems Agency. Galileo is the european global satellite-based navigation system [online]. *GSA Virtual Library, The*. (2016) [cited 2016-05-01]. Available from: http://www.gsa.europa.eu/galileo/why-galileo.

Feldmann, R. (2015). PSTAT 231: Data mining. Lecture notes at the University of California, Santa Barbara.

Glynn, E. F. Chromaticity diagrams [online]. (2009) [cited 2016-06-10]. Available from: http://www.efg2.com/Lab/Graphics/Colors/Chromaticity.htm.

GNSS Asia. India [online]. *GNSS.asia*. (2015) [cited 2016-05-14]. Available from: http://www.gnss.asia/india.

Gonzalez, R. C. and Woods, R. E. (2008a). *Digital Image Processing*, chapter 2, pages 52–58. PEARSON.

Gonzalez, R. C. and Woods, R. E. (2008b). *Digital Image Processing*, pages 861–907. PEARSON.

Gonzalez, R. C. and Woods, R. E. (2008c). *Digital Image Processing*, chapter 2, pages 35–46. PEARSON.

Gonzalez, R. C. and Woods, R. E. (2008d). *Digital Image Processing*, chapter 6, pages 394–460. PEARSON.

Gonzalez, R. C. and Woods, R. E. Digital image processing [online]. (2008). Available from: http://www.imageprocessingplace.com/DIP-3E/dip3e_book_images_downloads.htm.

Gonzalez, R. C. and Woods, R. E. (2008f). *Digital Image Processing*, chapter 6, pages 443–450. PEARSON.

Gonzalez, R. C. and Woods, R. E. (2008g). *Digital Image Processing*. PEARSON.

Gonzalez, R. C. and Woods, R. E. (2008h). *Digital Image Processing*, chapter 9, pages 627–680. PEARSON.

GPS.gov. Space segment [online]. (2016) [cited 2016-05-01]. Available from: http://www.gps.gov/systems/gps/space/#generations.

Hamiltion, Sir W. R. (1866). *Elements og Quaternions*. Longmans, Green, & Co.

Horn, B. K. (1987). Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642.

Horn, B. K., Hilden, H. M., and Negahdaripour, S. (1988). Closed-form solution of absolute orientation using orthonormal matrices. *JOSA A*, 5(7):1127–1135.

Kayser, K., Görler, J., Metze, K., Goldmann, T., Vollmer, E., Mireskandari, M., Kosjerina, Z., and Kayser, G. (2008). How to measure image quality in tissue-based diagnosis (diagnostic surgical pathology). *Diagnostic Pathology*, 3(11).

Kolodny, L. DroneDeploy Raises $2M to Make Drones Easy to Fly for Any Business [online]. *Venture Capital Dispatch*. (2014) [cited 2015-12-09]. Available from: http://blogs.wsj.com/venturecapital/2014/09/19/dronedeploy-raises-2m-to-make-drones-easy-to-fly-for-any-business/.

Kraus, K. (2007). *Photogrammetry - Geomatry from Images and Laser Scans*, volume 1. Walter de Gruyter, 2 edition.

Kreyzsig, E., Kreyszig, H., and Norminton, E. J. (2011). *Advanced Engineering Mathematics*, pages 322–354, 873–875. John Wiley & Sons, inc.

Kyriakidis, P. (2015). Analytical Methods in Geography III. Lecture notes at the University of California, Santa Barbara.

Land surveyors. History of land surveying [online]. (2010) [cited 2016-04-25]. Available from: http://www.landsurveyors.com/resources/history-of-land-surveying/.

Lay, D. C. (2012a). *Linear Algebra and Its Applications*, chapter 5, pages 265–327. Pearson, 4 edition.

Lay, D. C. (2012b). *Linear Algebra and Its Applications*, chapter 7, pages 414–423. Pearson, 4 edition.

Lay, D. C. (2012c). *Linear Algebra and Its Applications*, chapter 6, pages 338–344. Pearson, 4 edition.

Leica Geosystem Geospatial Imaging, LLC (2005). Imagine autosync™: Automated georeferencing for highly accurate data production. Available from: ftp://ftp.ecn.purdue.edu/jshan/proceedings/asprs2006/files/L1-5-0.pdf.

Li, Y. and Briggs, R. (2006). Automated georeferencing based on topological point pattern matching. In *The International Symposium on Automated Cartography (AutoCarto), Vancouver, WA*. Available from: http://www.cartogis.org/docs/proceedings/2006/li_briggs.pdf.

Ludwig, S., Nowak, M., Wilzbach, S., Cullen, C., mdondorff, JakobOvrum, Anderson, B., and Jost, D. Find, use and share dub packages [online]. *DUB - The D package regestry*. (2016) [cited 2016-05-20]. Available from: https://code.dlang.org/.

Mahalanobis, P. C. (1936). On the Generalized Distance in Statistics. *Proceedings of the National Institute of Science of India*, 12:49–55.

Massey, Jr, F. J. (1951). The kolmogoroc-smirnov test for goodness of fit. *Journal of the American Statistical Assosiation*, 46(253):68–78. Available from: http://www.jstor.org/stable/2280095.

MathWorks. Perform image processing, analysis, and algorithm development [online]. (2016) [cited 2016-05-20]. Available from: https://se.mathworks.com/products/image/?s_tid=srchtitle.

Nahavandchi, H. (2015). TBA4565: Geomatics, Specialization Course; GPS. Lecture notes at the Norwegian University of Science and Technoloy.

Nahavandchi, H., Haakonsen, T. A., and Aas, H. (2015). Accuracy investigations of uav photomapping over a test area in norway. *Kart og Plan*.

Nistad, S., BEGUERADJ, B., and Hexaholic. Opencv will not load a big image (4gb) [online]. (2016) [cited 2016-05-20]. Available from: http://stackoverflow.com/questions/35666761/opencv-will-not-load-a-big-image-4gb.

OpenDroneMap. Opendronemap [online]. (2015) [cited 20115-15-15]. Available from: https://github.com/OpenDroneMap/OpenDroneMap.

Orlov, A. Mahalanobis distance [online]. *Encyclopedia of Mathematics*. (2011) [cited 2016-05-05]. Available from: http://www.encyclopediaofmath.org/index.php?title=Mahalanobis_distance&oldid=17720.

Pix4D (2015). *Pix4D mapper manual*. Pix4D.

Ramebäck, C. (2003). Process automation systems - history and future. *Emerging Technologies and Factory Automation*, 1:3–4.

Rodges, C. (2010). Physics 1230: Light and Color. Lecture notes at the University of Colorado, Boulder.

Ronan, P. File:em spectrum.svg [online]. - *Wikimedia Commons*. (2007) [cited 2016-05-24]. Available from: https://commons.wikimedia.org/w/index.php?curid=2521356. Licensed under the CC BY-SA 3.0.

Rouse, M. Rgb (red, green, and blue) [online]. (2005) [cited 2016-04-27]. Available from: http://whatis.techtarget.com/definition/RGB-red-green-and-blue.

Rys, R. (2015). What is Lab color space? *HiDefColor*.

Skogseth, T. and Norberg, D. (1998a). *Grunnleggende Landmåling*, pages 9–13. Universitetsforlaget, Postboks 6860 St. Olavs plass, 0130 Oslo, 1 edition.

Skogseth, T. and Norberg, D. (1998b). *Grunnleggende Landmåling*. Universitetsforlaget, Postboks 6860 St. Olavs plass, 0130 Oslo, 1 edition.

Skogseth, T. and Norberg, D. (1998c). *Grunnleggende Landmåling*, pages 290–299. Universitetsforlaget, Postboks 6860 St. Olavs plass, 0130 Oslo, 1 edition.

startupticker.ch. Pix4d launches pix4dmapper [online]. *Start up ticker*. (2014) [cited 2015-12-09]. Available from: http://www.startupticker.ch/en/news/january-2014/pix4d-launches-pix4dmapper.

Theoharis, T., Papaioannou, G., Platis, N., and Patrikalakis, N. M. (2008). *Graphics & Visualization : Principles and Algorithms*, chapter 3, pages 108–114. A K Peters, Ltd.

Tipler, P. A. and Mosca, G. (2008). *Physics for Scientists and Engineers - with modern physics*, pages 1055–1096. W. H. Freeman and Company, W. H. Freeman and Comapny, 41 Madison Avenue, New York, 10010, 6 edition.

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (2000). Bundle adjustment — a modern synthesis. Available from: http://dx.doi.org/10.1007/3-540-44480-7_21.

Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(4):376–380.

van Helden, A. Galileo [online]. *Encyclopaedia Britannica. Britannica Academic*. (2016) [cited 2016-04-25]. Available from: http://academic.eb.com/EBchecked/topic/224058/Galileo.

Walpole, R. E., Myers, R. H., Myers, S. L., and Ye, K. (2012). *Probability & Statistics - for engineers and scientists*, pages 321–324. PEARSON, 9 edition.

Weisstein, E. W. Affine transformation. [online]. *MathWorld–A Wolfram Web Resource.* (2016) [cited 2016-04-28]. Available from: http://mathworld.wolfram.com/AffineTransformation.html.

Wicklin, R. What is the mahalanobis distance [online]. *DO Loop, The*. (2012) [cited 2016-05-05]. Available from: http://blogs.sas.com/content/iml/2012/02/15/what-is-mahalanobis-distance.html.

Zhang, M. Samsung 16TB SSD is the World's Largest Hard Drive [online]. *PetaPixel*. (2015) [cited 2016-01-02]. Available from: http://petapixel.com/2015/08/15/samsung-16tb-ssd-is-the-worlds-largest-hard-drive/.

Appendix A

Source Code

A.1 The program

Here is all the source code necessary to run the program in MATLAB®R2016a. In addition to MATLAB, the toolboxes "Image Processing Toolbox" version 9.4, "Statistics and Machine Learning Toolbox" version 10.2, and 'JSONlab'". The first two can be purchased by MathWorks, while the latter is freely available at `https://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab--a-toolbox-to-encode-decode-json-files`.

Listing A.1 is the main entry point of the application, and, for convenience, the first file of source code included. Afterwards follows the other custom m-scripts that are required by Listing A.1. They are listed alphabetically.

Figure A.1 shows how the different functions depend on each other.

Script A.1: main.m: The main entry point of the program.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [CM, ST, RMSE, mirrored] = main(orthophoto, gcp, varargin)
%% Discription
% MAIN is the main entery point for extracting ground controll points in
% the given ortophoto, and then georeference the image and rectify it.

%% Parse input

i_p = inputParser;
i_p.FunctionName = 'MAIN';

% Requiered
i_p.addRequired('orthophoto', @is_image_or_path);
i_p.addRequired('gcp', @is_point_list_or_path);
```

```
18
19 % Optional:
20 %Digital elevation model
21 i_p.addParameter('DigitalElevationModel', false, @is_dem_or_disabled);
22 i_p.addParameter('ElevationModel', false, @is_dem_or_disabled);
23 i_p.addParameter('DEM', false, @is_dem_or_disabled);
24
25 % Sample of ground control points
26 i_p.addParameter('GCPSample', '../../data/gcp_vals.mat',
       @is_sample_data_or_disabled);
27
28 % Area of a single ground control point in pixles.
29 i_p.addParameter('GCPArea', 9, @is_positive_integer);
30
31 % The maximum allowed root mean square error before mirroring the
32 % coordinates.
33 i_p.addParameter('MaxRMSE', 10^5, @is_positive_number);
34
35 % The distance function to be used.
36 i_p.addParameter('DistanceFunction', false, @is_function_or_logical);
37
38 % Max normalized distance. Used as a upper threshold.
39 i_p.addParameter('MaxDistance', 0.01, @is_number);
40
41 % Fraction of ground control points that need to match on the first run
42 i_p.addParameter('FractionMatched', 1/3, @is_fraction);
43
44 % Write world file
45 % Writes the wf iff the orthophoto was given by path.
46 i_p.addParameter('WriteWorldFile', true, @islogical);
47
48 % Rematch all points
49 i_p.addParameter('Rematch', true, @islogical);
50 % Which algorithm is to be used to find the absolute orientation?
51 i_p.addParameter('OrientationAlgorithm', 'Horn',
       @is_valid_orientation_algorithm);
52
53 i_p.addParameter('MinimumMatches', 5, @is_number);
54 i_p.addParameter('RadiusThreshold', 0.05, @is_number);
55 i_p.addParameter('AngleThreshold', 0.05, @is_number);
56
57 i_p.addParameter('UseProbability', true, @islogical);
58
59 i_p.addParameter('UseHornScale', false, @islogical);
60
61 i_p.parse(orthophoto, gcp, varargin{:});
62
63 %% Deal with the input
64 input = i_p.Results;
65
66 % Required
67 orthophoto      = input.orthophoto;
68 gcp             = input.gcp;
69
70 % Optional
71 dem             = get_dem(input);
72 gcp_sample      = input.GCPSample;
```

```
gcp_area         = input.GCPArea;
max_rmse         = input.MaxRMSE;
distance_fun     = input.DistanceFunction;
max_distamce     = input.MaxDistance;
min_fraction     = input.FractionMatched;
rematch          = input.Rematch;
use_probability  = input.UseProbability;
absor_alg        = input.OrientationAlgorithm;
min_matches      = input.MinimumMatches;
rad_thresh       = input.RadiusThreshold;
ang_thresh       = input.AngleThreshold;
use_horn_scale   = input.UseHornScale;

%% Initializing

% Orthophoto
orthophoto_path = '';
if ischar(orthophoto)
    orthophoto_path = orthophoto;
    orthophoto = im2double(imread(orthophoto));
elseif ~isa(orthophoto, 'double')
    orthophoto = im2double(orthophoto);
end
if size(orthophoto, 3) == 4
    % Remove trasnparencey / used it to mask the image
    orthophoto = orthophoto(:,:,1:3);
end

% Ground control points
if ischar(gcp)
    [gcp, crc, names] = load_geojson(gcp);
elseif is_point_list(gcp)
    warning('No datum of the coordinates was given!');
end

% Digitial elevation model
if ischar(dem)
    dem = imread(dem);
end

% Sample data of ground control points
if ischar(gcp_sample)
    gcp_sample = open(gcp_sample);
    fn = fieldnames(gcp_sample);
    gcp_sample = gcp_sample.(fn{1});
end
gcp_sample_lab = rgb2lab(gcp_sample);

if ~isa(distance_fun, 'function_handle')
    % Mahalanobis distance
    distance_fun = @(x) mahal_dist(x, gcp_sample_lab);
end

mirrored = false;

%% Get ground control points and match them
[location, ~, ~] = find_signal_colors(orthophoto, gcp_sample, '
```

```
    UseProbability', use_probability);

if dem
    heights = get_heights(dem, location, gcp_area);
    location = [location, heights];
end

min_matches = ceil(size(gcp, 1) * min_fraction);

[CM, ST, RMSE] = match_gcps(location, gcp, ...
    'GetOptimal', true, ...
    'ImageTPPMode', 'all', ...
    'MinimumMatches', min_matches, ...
    'OrientationAlgorithm', absor_alg, ...
    'UseHornScale', use_horn_scale, ...
    'AngleThreshold', ang_thresh, ...
    'RadiusThreshold' , rad_thresh);

%% Check if the points have to be mirrored
if RMSE > max_rmse
    location = mirror(location, 'Vertical');
    mirrored = true;
    % Alt. gcp = [gcp(:,2), gcp(:,1) gcp(:,3)];
    [CM, ST, RMSE] = match_gcps(location, gcp, ...
        'GetOptimal', true, ...
        'ImageTPPMode', 'all', ...
        'MinimumMatches', min_matches);
end

%% Find where all the ground control points are
if rematch
    [R_inv, t_inv, c_inv] = invert(ST);

    gcp_in_orthophoto = transform_points(gcp, R_inv, t_inv, c_inv);
    gcp_in_orthophoto = gcp_in_orthophoto(:,1:2);

    % Remove those that are outside of the image
    ortho_size = size(orthophoto);
    outside = gcp_in_orthophoto(:, 1) >  ortho_size(2) | ...
              gcp_in_orthophoto(:, 2) >  ortho_size(1) | ...
              gcp_in_orthophoto(:, 1) <= 0 | ...
              gcp_in_orthophoto(:, 2) <= 0;
    gcp_in_orthophoto(outside, :) = [];

    area = max([RMSE / 2, gcp_area * 4]);

    gcp_imgs = get_area(orthophoto, gcp_in_orthophoto, area);

    % Remove empty cells
    n_before = numel(gcp_imgs);
    gcp_imgs = remove_empty_cells(gcp_imgs);

    n_after = numel(gcp_imgs);

    if n_before ~= n_after
        warning('There are ground control points that falls outside the
    image. Consider increasing the minimum number of matched points.');
```

```
    end
    %% Get centres of each ground control point in images

    n = numel(gcp_imgs);
    image_coordinates = zeros(n, 2);
    for i = 1:n
        img = gcp_imgs{i};
        img = rgb2lab(img);
        distance_img = distance_fun(img);
        distance_img = normalize(distance_img);
        BW = distance_img <= max_distamce;
        image_coordinates(i, :) = get_centroid_of_largest_area( BW );
    end

    %% Calculate offsets
    centre_of_images = ceil(size(gcp_imgs{1}) / 2);
    centre_of_images = centre_of_images(1:2);

    offset = bsxfun(@minus, image_coordinates, centre_of_images);

    location = round(gcp_in_orthophoto + offset);

    %% Rematch all points
    if dem
        heights = get_heights(dem, location, gcp_area);
        location = [location, heights];
    end

    [CM, ST, RMSE] = match_gcps(location, gcp, ...
        'GetOptimal', true, ...
        'ImageTPPMode', 'all', 'MinimumMatches', size(location, 1));
end

if ~strcmp(orthophoto_path, '')
    C = strsplit(orthophoto_path, '.');
    ext = C{end};
    ext = strcat(ext(1), 'wf');
    path = cell((numel(C) - 1) * 2, 1);
    for i = 1:2:2*(numel(C) - 1)
        path{i} = C{i};
        path{i + 1} = '.';
    end
    path = strjoin(path, '');
    worldfile_path = strcat(path, ext);

    write_world_file(worldfile_path, ST);
end

end

%==========================================================================
%% ADDITIONAL FUNCTIONS
%==========================================================================

%% GET_DEM
%==========================================================================
function res = get_dem( input )
```

```
242
243 if input.DEM
244     res = input.DEM;
245 elseif input.ElevationModel
246     res = input.ElevationModel;
247 elseif input.DigitalElevationModel
248     res = input.DigitalElevationModel;
249 else
250     res = false;
251 end
252
253 end
254
255 %% GET_CENTROID_OF_LARGEST_AREA
256 %=========================================================================
257 function res = get_centroid_of_largest_area( BW )
258
259 stats = regionprops(BW, 'Area', 'Centroid');
260
261 n = numel(stats);
262
263 if n == 0
264     % There are no areas
265     warning('One of the ground control points have been placed outside the
          image.');
266     res = ceil(size(BW));
267     return
268 end
269
270 areas = zeros(n, 1);
271
272 for i = 1:n
273     stat = stats(i);
274     areas(i) = stat.Area;
275 end
276
277 [~, I] = max(areas);
278 biggest = stats(I);
279
280 res = biggest.Centroid;
281
282 end
```

Script A.2: apply_fun2img.m: Applies a given function to all pixels of an image.

```
1  % This Source Code Form is subject to the terms of the Mozilla Public
2  % License, v. 2.0. If a copy of the MPL was not distributed with this
3  % file, You can obtain one at http://mozilla.org/MPL/2.0/.
4  %            Copyright (c) 2016 Sindre Nistad
5  function res = apply_fun2img(fun, image)
6  %% Discription
7  % APPLY_FUN2IMG applies the given function to each color of the given
8  % image, if the image has 3 bands, otherwise, it is assumed that the
9  % function takes a single column vector as input that has the same size as
10 % the number of bands.
11
12 %% Error checking
```

```
if ~isa(fun, 'function_handle')
    error('The given function is not a function handle');
end

%% Initializing
dims = size(image);

%% Apply function
res = fun(reshape(image, [dims(1) * dims(2), dims(3)]));
res = reshape(res, [dims(1), dims(2)]);

end
```

Script A.3: bounding_box2area.m: Calculates the area of a bounding box. The format can be MATLAB's format, or [min_x, max_x, min_y, max_y].

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function area = bounding_box2area( BB )
%% BOUNDING_BOX2AREA: Calculates the area of a given bounding box

%% Initializing
if iscell(BB)
    BB = BB{:};
end

[x_min, x_max, y_min, y_max] = bounding_box2limits(BB);


%% Calculate area

area = (x_max - x_min + 1) * (y_max - y_min + 1);

end
```

Script A.4: bounding_box2limits.m: Converts a bounding box, as returned from the built-in function `regionprops` to the extents; [min_x, max_x, min_y, max_y].

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [x_min, x_max, y_min, y_max] = bounding_box2limits(BB)
%% Discription
% A function that converst the BoundingBox given by regionprops to the
% vector [x_min, x_max, y_min, y_max].
% The output can be a matrix, a vector, or 4 scalars

%% Type checking
if isstruct(BB) && numel(BB) == 1
    BB = BB.BoundingBox;
elseif isstruct(BB)
    error('There are multiple bounding boxes.');
end

```

```
%% Extracting values
x_start = BB(1);  % double (start.5000)
y_start = BB(2);  % double (start.5000)

width   = BB(3);  % int
height  = BB(4);  % int

%% Computing the limits
x_min = ceil(x_start);
y_min = ceil(y_start);

x_max = x_min + width - 1;   % The width includes the start
y_max = y_min + height - 1;  % The height includes the start

%% Generating outputs
if nargout <= 1 && numel(BB) == 1
    x_min = [x_min, x_max, y_min, y_max];
end
```

Script A.5: bounding_box2points.m: Converts a bounding box, as defined by the extents of each axis to a list of points that represents the extents of the bounding box.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function BB_points = bounding_box2points(BB)
%% Description
% A simple function that converts a list of bounding boxes that are
    defined
% as [x_min x_max y_min y_max] to a list of points of the form
% x_min y_min
% x_min y_max
% x_max y_min
% x_max y_max

%%
BB_points = [BB(:,1) BB(:,3) ;
             BB(:,1) BB(:,4) ;
             BB(:,2) BB(:,3) ;
             BB(:,2) BB(:, 4)];
end
```

Script A.6: create_mask.m: Creates a binary image, or mask, based on given limits for each color band.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function BW = create_mask(img, limits)
%% Discription
% Creates a mask of the image based on the limits given for each band. It
% is in the form:
% [min_band_1, max_band_1
%  min_band_2, max_band_2
%            .
```

```
%            .
%            .
%  min_band_n, max_band_n]
% It is not neccessary that they are ordered by minimun first, and then
% maximum, but the bands must be in that order.

%% Initializing
img_size = size(img);
num_bands = img_size(3);
BW = ones(img_size(1:2));

%% Creating the mask, band for band
for i = 1:num_bands
    min_val = min(limits(i, :));
    max_val = max(limits(i, :));
    BW = BW & min_val <= img(:,:, i) & img(:,:,i) <= max_val;
end

end
```

Script A.7: divide_image_into_bounding_boxes.m: Takes an image, and a mask, or binary image, and gives a cell-array of smaller images whose extent is the same as the bounding box of each separate object in the mask. Useful to drastically reduce the computational time, and memory needed.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [images, locations] = divide_image_into_bounding_boxes(img, BW)
%% Discription
% This function divides an image up into ounding boxes of the areas as
% specified by the binary image BW.
% If only a binary image is given, images will consist of binary images.
% The function also returns the locations of the different images. This is
% done by giving the respective bounding boxes as a matrix, 'locations',
% whose rows are vectors of the form [x_min, x_max, y_min, y_max].
%
% Usage:
% [images, locations] = divide_image_into_bounding_boxes(img, BW)
% [images, locations] = divide_image_into_bounding_boxes(BW)
%

%% Check consistencies
if nargin == 2
    img_size = size(img);
    BW_size = size(BW);

    if img_size(1:2) ~= BW_size
        error('The image and the mask have different sizes');
    end
elseif nargin == 1
    % Matlab hack
    BW = img;
end

%% Extracting bounding boxes
res = regionprops(BW, 'BoundingBox', 'PixelList');
```

```
num_obj = numel(res);

images = cell(num_obj, 1);
locations = zeros(num_obj, 4);

for i=1:num_obj
    element = res(i);
    boundingBox = element.BoundingBox;

    [x_min, x_max, y_min, y_max] = bounding_box2limits(boundingBox);

    if nargin == 2
        % Matices are indexed col, row
        images(i) = {img(y_min:y_max, x_min:x_max, :)};
    else
        images(i) = {BW(y_min:y_max, x_min:x_max)};
    end
    locations(i, :) = [x_min, x_max, y_min, y_max];
end

end
```

Script A.8: extract_parameters_from_similarity_transform.m: Extracts the rotation matrix, $\mathbf{R}$, the translation, $\boldsymbol{t}$, and the scale factor c from a Similarity Transform object, as returned by the absolute orientation functions.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [R, t, s] = extract_parameters_from_similarity_transform( ST )
%% Discription
% EXTRACT_PARAMETERS_FROM_SIMILARITY_TRANSFORM extracts the rotational
% matrix R, the translation vector t, and the scale factor s from the
% similarity transform, ST, cell array.
% Usefull when dealing with the results from match_gcps.m, and others.

%% Extraction
R = ST{1};
t = ST{2};
s = ST{3};

end
```

Script A.9: find_signal_colors.m: detects the location of candidates for the ground control points (GCPs). In addition to these positions, it also gives an estimate of the probability of each of the candidates. The output is sorted so that the most likely candidates come first.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [location, probability, imgs] = find_signal_colors(image,
    gcp_data, varargin)
%% Discription
% A function that finds candidates for ground control points that have a
% signal color (orange), and an estimate of how likely each point is to be
```

```
% a ground control point. The lists are sorted byr probability in
    decending
% order, i.e. the most likely candidate is first and the least likely
% candidate is last.

%% Parse input

i_p = inputParser;
i_p.FunctionName = 'FIND_SIGNAL_COLOR';

% Requiered
i_p.addRequired('image', @is_image_or_path);
i_p.addRequired('gcp_data', @is_point_list);

% Optional:
% Method to be used to reduce data
i_p.addParameter('LimitsMode', 'min-max', @is_min_max_std_mean);
% What is to be used with points that are too close to eachother?
% remove, average or probable.
i_p.addParameter('ReplacePointsTooClose', 'probable', @is_replace_mode);
% Toggle whether or not to use a median filter
i_p.addParameter('MedianFilter', false, @islogical);
% Toggle whether or not areas that are of size [1 x n] or [n x 1] is to be
    removed.
i_p.addParameter('VectorFilter', true, @islogical);
i_p.addParameter('UseMorphology', true, @islogical);
% How large does an area have to be before it is considered?
% Areas less than or equal to this value will be removed. If strictly
% larger than 1, no areas will be removed.
i_p.addParameter('MinmumArea', 0, @is_number);
% The (totatl) procentile to be GCP samples to be discarted when
% initial rough selection is done.
i_p.addParameter('PercentileRoughSelection', 0.25, @is_fraction);
% The wheight given to the standard deviation when selecting a quantile
% for the samples to be ignored when the rough selection is done.
i_p.addParameter('StandardDeviationWeight', 1, @is_number);
% This sais how much of an image can have 0 probability.
i_p.addParameter('ProbabilityPortion', 0.25, @is_fraction);
% Used to eliminate images that have too much clutter.
i_p.addParameter('MaximumClutter', 2, @is_number);
% The minimum allowed likelyhood for an area to be considered probable.
i_p.addParameter('MinimumProbability', 10^-07, @is_fraction);
% This is the smallest side length that is allowed for a sub_image.
i_p.addParameter('MinimumSideLength', 3, @is_integer);
% The minimum distance allowed between two points
i_p.addParameter('MinimumDistanceBetweenPoints', 30, @is_number);
% The disatance in each direction from the centre of a candidate point
% that is sampled in order to determin if it is a homogenious area.
i_p.addParameter('SampleArea', 40, @is_integer);
% Specifies the maximum allowed standard deviation of a sub image.
i_p.addParameter('MaximumSpread', 0.03, @is_number);
% The maximum squared distance for a color to be away from the
% signal color sample. Max distance using emperical values:
% RGB: 19.821439604735122, Lab: 24.060660742824890.
i_p.addParameter('MaximumRoughDistance', 20, @is_number);
i_p.addParameter('MaximumFineDistance', 6, @is_number);
% Toggles whether the image is to be sharpened before processed.
```

```
64 i_p.addParameter('SharpenImage', false, @islogical);
65
66 i_p.addParameter('StructureElement', strel('Disk', 10),
       @is_structure_element);
67 % Toggles whether probability is to be used. If gcp_sample is very large,
68 % turning it on may cause the program to be VERY slow.
69 i_p.addParameter('UseProbability', true, @islogical);
70 % By what means are the center of the candidate GCP chosen?
71 i_p.addParameter('SelectCenter', 'Centroid', @is_center);
72
73 i_p.addParameter('MinimumMatches', -1, @is_number);
74
75 i_p.parse(image, gcp_data, varargin{:});
76
77 %% Deal with the input
78 input = i_p.Results;
79
80 % Required
81 image                       = input.image;
82 gcp_data                    = input.gcp_data;
83
84 % Optional
85 limits_method               = input.LimitsMode;
86 points_too_close            = input.ReplacePointsTooClose;
87 use_median                  = input.MedianFilter;
88 remove_vectors              = input.VectorFilter;
89 use_morphology              = input.UseMorphology;
90 % Internal parameters
91 remove_areas_leq            = input.MinmumArea;
92 percentile_rough_selection  = input.PercentileRoughSelection / 2;
93 std_weight                  = input.StandardDeviationWeight;
94 zero_probability_ratio      = input.ProbabilityPortion;
95 max_elements                = input.MaximumClutter;
96 min_probability             = input.MinimumProbability;
97 min_side                    = input.MinimumSideLength;
98 min_distance                = input.MinimumDistanceBetweenPoints;
99 sample_area                 = input.SampleArea;
100 max_spread                  = input.MaximumSpread;
101 max_rough_distance          = input.MaximumRoughDistance;
102 max_fine_distance           = input.MaximumFineDistance;
103 sharpen                     = input.SharpenImage;
104 SE                          = input.StructureElement;
105 use_probability             = input.UseProbability;
106 select_center               = input.SelectCenter;
107
108 %% Initializing
109 if ~isa(image, 'double')
110     image = im2double(image);
111 end
112
113 if sharpen
114     image = imsharpen(image);
115 end
116
117 if ~use_probability
118     % The default option for removing points that
119     % are too close to each other, is incopatible
```

```
120     % with not using probability.
121     points_too_close = 'average';
122     warning('The option probable for ''ReplacePointsTooClose'' is
        incompatible ''UseProbability'' set to false.');
123 end
124
125 % Convert data to Lab
126 gcp_lab = rgb2lab(gcp_data);
127
128 % Distance metric
129 % RGB
130 C = cov(gcp_data)\eye(size(gcp_data, 2));
131 m = mean(gcp_data);
132 distance_metric = @(x) mahal_dist(x, C, m);
133 % Lab
134 C = cov(gcp_lab)\eye(size(gcp_lab, 2));
135 m = mean(gcp_lab);
136 distance_metric_lab = @(x) mahal_dist(x, C, m);
137
138 % Probability distributions
139 if use_probability
140     [PD_L, PD_a, PD_b] = get_most_likely_distribution(gcp_lab);
141     pd_L = get_pdf(PD_L); pd_a = get_pdf(PD_a); pd_b = get_pdf(PD_b);
142     prob_dist = @(x) pd_L(x(:, 1)) .* pd_a(x(:, 2)) .* pd_b(x(:, 3));
143     aggregate_probability = @(img) min(quantile(img(:), 0.95));
144 end
145 spread_function = @(img) std(img(:));
146
147 %% Reducing the amount of data
148
149 % Getting limits of color for the signal colors
150 if strcmp(limits_method, 'min-max')
151     lim = limits(gcp_data, limits_method, percentile_rough_selection);
152 elseif strcmp(limits_method, 'mean-std')
153     lim = limits(gcp_data, limits_method, std_weight);
154 else
155     lim = limits(gcp_data);
156 end
157
158 % Creating a binary image of the
159 BW = create_mask(image, lim);
160
161 if use_median
162     BW = medfilt2(BW, 'symmetric');
163 end
164
165 if remove_areas_leq > 0
166     BW = remove_areas(BW, 'Area', [remove_areas_leq, Inf], 'MinBoundary',
        'Exclusive');
167 end
168
169
170 %% Split data into smaler chuncks
171 [images, locations] = divide_image_into_bounding_boxes(image, BW);
172 n_img = numel(images);
173 if remove_vectors
174     too_small = false(n_img, 1);  % Instead of logical(zeros)
```

```
    for i=1:n_img
        if size(images{i}, 1) < min_side || size(images{i}, 2) < min_side
            too_small(i) = true;
        end
    end
    images = images(~too_small);
    locations = locations(~too_small, :);
end

%% Finer selection
[candidate_images, distance_images] = threshold_distance(images,
    distance_metric, max_rough_distance);
images = images(candidate_images);
locations = locations(candidate_images, :);

%% Remove morthological incorrect images
right_morphology = prune_morphology(images, distance_metric,
    max_rough_distance, 'Area', true, 'Eccentricity', true, 'AreaPerimeter
    ', true);

% Remove wrong images
distance_images = distance_images(right_morphology);
images = images(right_morphology);
locations = locations(right_morphology, :);


%% Even finer selection
% Uses Lab color space, and takes some of the area round the image intop
% account.
lab_images = convert_elements(images, @rgb2lab);
[candidate_images, distance_images] = threshold_distance(lab_images,
    distance_metric_lab, max_fine_distance);

images = images(candidate_images);
lab_images = lab_images(candidate_images);
locations = locations(candidate_images, :);

large_images = get_area(image, locations, sample_area);
large_images = convert_elements(large_images, @rgb2lab);
right_morphology = prune_morphology(large_images, distance_metric_lab,
    max_fine_distance, 'Eccentricity', true, 'Area', true, 'AreaPerimeter'
    , true, 'Tightness', true, 'Median', true, 'Fill', true);

images = images(right_morphology);
lab_images = lab_images(right_morphology);
locations = locations(right_morphology, :);
large_images = large_images(right_morphology);
distance_images = distance_images(right_morphology);

%% Calculate probabilities
if use_probability
    n_img = numel(images);

    prob_imgs = cell(n_img, 1);

    for i = 1:n_img
        prob_imgs{i} = apply_fun2img(prob_dist, lab_images{i});
```

```
    end
end


%% Remove the (impossibly) unlikely candidates
if use_probability
    n_img = numel(images);
    no_chance = false(n_img, 1);
    for i = 1:n_img
        prob_img = prob_imgs{i};
        if max(max(prob_img)) < min_probability
            no_chance(i) = true;
        else
            area = prod(size(prob_img));
            zero_chance = sum(sum(prob_img == 0));
            if zero_chance / area >= zero_probability_ratio
                no_chance(i) = true;
            end
        end
    end

    % Remove the unlikely candidates
    distance_images = distance_images(~no_chance);
    images = images(~no_chance);
    lab_images = lab_images(~no_chance);
    locations = locations(~no_chance, :);
    prob_imgs = prob_imgs(~no_chance);
    large_images = large_images(~no_chance);

end

%% Calculate probabilities
if use_probability
    n_img = numel(images);
    probability = zeros(n_img, 1);
    for i = 1:n_img
        prob_img = prob_imgs{i};
        probability(i) = aggregate_probability(prob_img);
    end
end

%% Compute output
n_img = numel(images);

location = zeros(n_img, 2);
% This can be done earlier; when we look for
% images that have morphological errors
if strcmpi(select_center, 'Probability')
    probability = zeros(n_img, 1);
    for i = 1:n_img
        prob_img = prob_imgs{i};
        [row, col] = find(prob_img == max(max(prob_img)));
        if numel(row) >= 2 || numel(col) >= 2
            row = round(mean(row));
            col = round(mean(col));
        end
        bounding_box = locations(i, :);
```

```
283         x_min = bounding_box(1);
284         y_min = bounding_box(3);
285         location(i, :) = [x_min + col, y_min + row];
286         probability(i) = aggregate_probability(prob_img);
287     end
288 elseif strcmpi(select_center, 'Mahalanobis')
289     for i = 1:n_img
290         dist_img = distance_images{i};
291         [row, col] = find(dist_img == min(dist_img(:)));
292         if numel(row) >= 2 || numel(col) >= 2
293             row = round(mean(row));
294             col = round(mean(col));
295         end
296         bounding_box = locations(i, :);
297         x_min = bounding_box(1);
298         y_min = bounding_box(3);
299         location(i, :) = [x_min + col, y_min + row];
300     end
301 elseif strcmpi(select_center, 'Centroid')
302     for i = 1:n_img
303         BW = distance_images{i} <= max_fine_distance;
304         res = regionprops(BW, 'Centroid');
305         centroid = res.Centroid;
306         col = round(centroid(1)); row = round(centroid(2));
307         bounding_box = locations(i, :);
308         x_min = bounding_box(1);
309         y_min = bounding_box(3);
310         location(i, :) = [x_min + col, y_min + row];
311     end
312 end
313
314 %% Deal with points that are too close
315 n_img = numel(images);
316
317 Z = squareform(pdist(location));
318 if strcmp(points_too_close, 'off')
319     % Do nothing
320 elseif strcmp(points_too_close, 'remove')
321     [row, col] = find(Z < min_distance & Z ~= 0);
322 elseif strcmp(points_too_close, 'average')
323     row = false(n_img, 1);
324     for i = 1:n_img
325         if row(i) == 1
326             % We have already decided to remove this
327             continue
328         end
329         too_close = find(Z(i, :) < min_distance & Z(i, :) ~= 0);
330         if numel(too_close) > 0
331             too_close = [too_close i];
332             mean_position = mean(location(too_close, :));
333             D = sqrt((bsxfun(@minus, location(too_close, :), mean_position
    )).^2);
334             min_D = min(D);
335             keep = find(D(:, 1) == min_D(1) & D(:, 2) == min_D(2));
336             if numel(keep) > 1
337                 keep = keep(1);  % In case multiple are as close.
338             end
```

```
            row(too_close) = true;
            row(too_close(keep)) = false;
        end
    end
elseif strcmp(points_too_close, 'probable')
    row = false(n_img, 1);
    for i = 1:n_img
        if row(i) == 1
            % We have already decided to remove this
            continue
        end
        too_close = find(Z(i, :) < min_distance & Z(i, :) ~= 0);
        if numel(too_close) > 0
            too_close = [too_close i];
            keep = find(probability == max(probability(too_close)));
            if numel(keep) > 1
                keep = keep(1);  % In case multiple are as close.
            end
            row(too_close) = true;
            row(keep) = false;
        end
    end
end

distance_images(row) = [];
images(row) = [];
location(row, :) = [];
if use_probability
    prob_imgs(row) = [];
    probability(row) = [];
end
large_images(row) = [];

%% Calculate how "close" the areas are to a circle
n_img = numel(images);
closeness = zeros(n_img, 1);

for i = 1:n_img
    img = large_images{i};
    BW = distance_metric_lab(img) < max_fine_distance;
    BW = imclose(BW, SE);
    BW = imfill(BW, 'Holes');
    stats = regionprops(BW, 'Area', 'BoundingBox');
    n_stats = numel(stats);
    areas = zeros(n_stats, 1);
    temp_closeness = zeros(n_stats, 1);
    for j = 1:n_stats
        stat = stats(j);
        region_area = stat.Area;
        bounding_box = stat.BoundingBox;
        [x_min, x_max, y_min, y_max] = bounding_box2limits(bounding_box);
        width = x_max - x_min + 1;
        height = y_max - y_min + 1;
        side = max([width, height]);

        areas(j) = region_area;
        temp_closeness(j) = abs(region_area / side^2 - pi / 4);
```

```
396     end
397     [~, I] = max(areas);  % Ascending order
398     closeness(i) = temp_closeness(I(end));
399 end
400 
401 [~, I] = sort(closeness);
402 
403 %% Sort the candidates
404 %[probability, I] = sort(probability, 'descend');
405 if use_probability
406     probability = probability(I);
407 else
408     probability = 0;
409 end
410 location = location(I, :);
411 
412 imgs = images(I);
413 
414 end
415 
416 %% Additional functions
417 
418 %%
419 %=========================================================================
420 function [within, distimgs] = threshold_distance(images, distance_metric,
        threshold)
421 n_img = numel(images);
422 candidate_images = false(n_img, 1);
423 distance_images = cell(n_img, 1);
424 
425 for i = 1:numel(images)
426     image_part = images{i};
427     dist_part = distance_metric(image_part);
428     BW_part = dist_part <= threshold;
429     BW_part = medfilt2(BW_part, 'symmetric');  % Removes images that only
        have a pixel or two.
430     if any(any(BW_part))
431         candidate_images(i) = true;
432         distance_images{i} = dist_part;
433     end
434 end
435 
436 % Remove empty cells
437 distimgs = remove_empty_cells(distance_images);
438 within = candidate_images;
439 end
440 
441 %%
442 %=========================================================================
443 function [PD_band1, PD_band2, PD_band3] = get_most_likely_distribution(
        data)
444 
445     PD_band1 = fitdist(data(:, 1), 'Kernel', 'Kernel', 'Normal');
446 
447     PD_band2 = fitdist(data(:, 2), 'Kernel', 'Kernel', 'Normal');
448 
449     PD_band3 = fitdist(data(:, 3), 'Kernel', 'Kernel', 'Normal');
```

```
end

%%
%=========================================================================
function cellarray = convert_elements(images, fun)
    n = numel(images);
    cellarray = cell(n, 1);

    for i = 1:n
        e = images{i};
        cellarray{i} = fun(e);
    end
end

%%
%=========================================================================
function output_arg = is_center( input_arg )
%% Discription
% IS_CENTER checks if the given argument is a valid method of selecting
% the center of a candidate GCP.

%% Check
output_arg = strcmpi(input_arg, 'Centroid') || ...
             strcmpi(input_arg, 'Probability') || ...
             strcmpi(input_arg, 'Mahalanobis');

end
```

Script A.10: get_area: Extracts an area around a given point of a given image. A user, or program can specify the dimension of the area to be returned.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function imgs = get_area(image, locations, area)
%% Discription
% GET_AREA extracts an area around the points in location from the given
% image. The size of the area is given by *area*. The point in location
% will then be in the middle of the area.

%% Initialization
if iscell(image)
    error('The image must be the original image, and NOT a set of images')
    ;
end

if area < 0
    area = 0;
end

image_size = size(image);
n = size(locations, 1);
imgs = cell(n, 1);

%% Extracting areas
for i = 1:n
```

```
    [x, y] = extract_location(image_size, locations(i, :), area);
    imgs{i} = image(y, x, :);
end

if n == 1
    imgs = imgs{1};
end

end

%=========================================================================
%% ADDITIONAL FUNCTIONS
%=========================================================================

%% EXTRACT_LOCATION
%=========================================================================
function [x, y] = extract_location(image_size, location, area)
%% Calculating centrum of area

location = round(location);
area = round(area);

if numel(location) == 2 || numel(location) == 3
    x = location(1) - area : location(1) + area;
    y = location(2) - area : location(2) + area;
elseif numel(location) == 4
    x_centrum = floor((location(2) - location(1)) / 2) + location(1);
    y_centrum = floor((location(4) - location(3)) / 2) + location(3);

    x = x_centrum - area : x_centrum + area;
    y = y_centrum - area : y_centrum + area;
else
    error('The numer of elements in location is wrong. It must be 2 or 4')
    ;
end

%% Error checking
if max(x) > image_size(2)
    x = min(x) : image_size(2);
end
if max(y) > image_size(1)
    y = min(y) : image_size(1);
end
if min(x) < 1
    x = 1 : max(x);
end
if min(y) < 1
    y = 1 : max(y);
end

end
```

Script A.11: get_heights.m: Extracts the heights at certain locations and samples a given area around the location. Returns the average elevation of the area.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
```

```
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function heights = get_heights( dem, points, area )
%% Discription
% GET_HEIGHS takes a digital elevation model and a set of points as input,
% and gives out the heights of each point in order. Area is an optinal
% argument that specifies the side length of the sample area for each
% point. The defalut is 1.
% The dem argument can be either the full elevation model, or it can be
    the
% path to the model.

%% Initiliazation
if nargin == 2
    area = 1;
end

if ischar(dem)
    dem = imread(dem);
end

n = size(points, 1);
heights = zeros(n, 1);

%%

height_cells = get_area(dem, points, round(sqrt(area) / 2 - 1));
for i = 1:n
    height_area = height_cells{i};
    heights(i) = mean(height_area(:));
end

end
```

Script A.12: get_pdf.m: Extracts the i^{th} component of a probability density function, and the corresponding cumulative probability function. Useful when dealing with cell-arrays of probability distribution objects.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [pdf, cdf] = get_pdf(PD, i)
%% Discription
% A function to extract the i-th probability denisity function and the
% cummulative distributiuon function of the collection PD. If i is not
% given, the pdf will be extracted from PD, as it is assumed that PD is a
% single element.

%% Check number of parameters
if nargin == 2
    Dist = PD(i);
else
    Dist = PD;
end

%% Get the probability density function
```

```
if isstruct(Dist)
    params = num2cell(Dist.Params);
    Dist = makedist(Dist.DistName, params{:});
end
pdf = @(x) Dist.pdf(x);
cdf = @(x) Dist.cdf(x);

end
```

Script A.13: horn.m: An implementation of Horn (1987). It is an algorithm for computing the absolute orientation with a least-squares estimation (LSE) technique. It uses the concept of quaternions for computing the rotation matrix $\mathbf{R}$.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [R, t, s, RMSE] = horn(A, B)
%% Discription
% HORN implements the algorithm for absolute orientation given by Berthold
% K. P. Horn in the article "Closed-form solution of absolute orientation
% using unit quaternions" (1986).
% The input is two sets of points, *A*, and *B*. *A* is the set of point
% that is to be transformed into the same coordinate system as the points
% in *B*.
% The output is the rotation matrix *R*, the translation *t*, and the
    scale
% factor *s*.
%

%% Initialization

dim = min([size(A, 2), size(B, 2)]);
A = A(:, 1:dim);
B = B(:, 1:dim);

r_to = bsxfun(@minus, B, mean(B));      % Equivilent to r_r
r_from = bsxfun(@minus, A, mean(A));    % Equivilent to r_l

norm = @(x) dot(x, x, 2);

%% Scale factor

s = sqrt(sum(norm(r_to)) / sum(norm(r_from)) );

%% Rotation

M = r_from' * r_to;
if all(size(M) == [2 2])
    tmp = zeros(3);
    tmp(1:2,1:2) = M;
    M = tmp;
end

S_xx = M(1, 1); S_xy = M(1, 2); S_xz = M(1, 3);
S_yx = M(2, 1); S_yy = M(2, 2); S_yz = M(2, 3);
S_zx = M(3, 1); S_zy = M(3, 2); S_zz = M(3, 3);
```

```
44
45 N = [S_xx + S_yy + S_zz          S_yz - S_zy          S_zx - S_xz
   S_xy - S_yx; ...
46         S_yz - S_zy          S_xx - S_yy - S_zz       S_xy + S_yx
   S_zx + S_xz; ...
47         S_zx - S_xz              S_xy + S_yx      -S_xx + S_yy - S_zz
   S_yz + S_zy;...
48         S_xy - S_yx              S_zx + S_xz          S_yz + S_zy      -S_xx
   - S_yy - S_zz];
49
50 [V, D] = eig(N);
51 e = D(1:size(D, 1) + 1: end);  % Extract the eigen values.
52
53 [~, I] = max(e);
54 q = V(:, I);
55
56 R = quat2rotmat(q);
57 R = R(1:dim, 1:dim);
58
59 %% Translation
60
61 t = bsxfun(@minus, mean(B)', s * R * mean(A)');
62
63 %% Root mean square error
64
65 RMSE = rmse(B, A, @(x) transform_points(x, R, t, s));
66
67 end
68
69 function R = quat2rotmat(q)
70 q_0 = q(1); q_x = q(2); q_y = q(3); q_z = q(4);
71
72 R = [q_0^2 + q_x^2 - q_y^2 - q_z^2         2* (q_x *q_y - q_0 * q_z)      2
   * (q_x * q_z + q_0 * q_y); ...
73       2 * (q_y * q_x + q_0 * q_z)        q_0^2 - q_x^2 + q_y^2 - q_z^2   2
   * (q_y * q_z - q_0 * q_x); ...
74       2 * (q_z * q_x - q_0 * q_y)        2 * (q_z * q_y + q_0 * q_x)
       q_0^2 - q_x^2 - q_y^2 + q_z^2];
75 % Q = [q_0 -q_x -q_y -q_z; ...
76 %       q_x  q_0 -q_z  q_y;...
77 %       q_y  q_z  q_0 -q_x;...
78 %       q_z -q_y  q_x  q_0];
79 %
80 %  Q_bar = [q_0 -q_x -q_y -q_z; ...
81 %            q_x  q_0  q_z -q_y;...
82 %            q_y -q_z  q_0  q_x;...
83 %            q_z  q_y -q_x  q_0];
84 % R = Q_bar' * Q;
85 % R = R(2:end, 2:end);
86 end
```

Script A.14: horn_hilden.m: An implementation of Horn et al. (1988). It is an algorithm for computing the absolute orientation with a LSE technique. It uses an orthonormal matrix for computing the rotation matrix $\mathbf{R}$.

```
1 % This Source Code Form is subject to the terms of the Mozilla Public
2 % License, v. 2.0. If a copy of the MPL was not distributed with this
```

```
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [R, t, s, RMSE] = horn_hilden(A, B)
%% Discription
%

%% Initilaization
if size(A, 2) == 2
    A = [A zeros(size(A, 1), 1)];
end

if size(B, 2) == 2
    B = [B zeros(size(B, 1), 1)];
end

%% Scale factor
r_from_mean = mean(A);
r_to_mean   = mean(B);

r_from      = bsxfun(@minus, A, r_from_mean);
r_to        = bsxfun(@minus, B, r_to_mean);

s = sqrt( sum(dot(r_to, r_to, 2)) / sum(dot(r_from, r_from, 2)) );

%% Rotaion

M = r_from' * r_to;

%S = (M' * M)^(1/2);

if rank(M) == 2
    [V, D] = eig(M' * M);
    e = diag(D);

    [~, I] = sort(abs(e), 'descend');
    e_1 = e(I(1)); e_2 = e(I(2));
    u_1 = V(:,I(1)); u_2 = V(:,I(2));
    S_plus = 1/sqrt(e_1) .* u_1 * u_1' + 1/sqrt(e_2) .* u_2 * u_2';
    [U_0, ~, V_0] = svd(M * S_plus);
    u_3 = U_0(:, 3); v_3 = V_0(:, 3);

    R = M * S_plus + u_3 * v_3';
    if sign(det(R)) == -1
        R = M * S_plus - u_3 * v_3';
    end
else
    R = (M / (sqrtm(M' * M)))';
end
%% Translation
t = r_to_mean' - s * R * r_from_mean';

%% Root mean square error
RMSE = rmse(B, A, @(x) transform_points(x, R, t, s));

end
```

Script A.15: invert.m: Inverts the absolute orientation parameters, similarity transform (ST). Then one has a transformation from a real world coordinate system to image coordinates. Useful when checking where the rest of the GCPs are in the image.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function varargout = invert(varargin)
%% Discription
% INVERT inverts the the rotation matrix, translation and the scaling
% factor. It can also take a set of similarity transform, ST = {R, t, s}.
%
% Uses:
% ST = invert(ST)
% ST = invert(R, t, s)
% [R, t, s] = invert(ST)
% [R, t, s] = invert(R, t, s)

%% Initialization
if nargin == 1
    ST = varargin{1};
    R = ST{1};
    s = ST{3};
    t = ST{2};
elseif nargin == 3
    R = varargin{1};
    t = varargin{2};
    s = varargin{3};
end

%% Inversion

R_inv = R \ eye(size(R));
c_inv = 1 / s;
t_inv = - 1 / s * R_inv * t;

if nargout <= 1
    varargout{1} = {R_inv, t_inv, c_inv};
else
    varargout{1} = R_inv;
    varargout{2} = t_inv;
    varargout{3} = c_inv;
end

end
```

Script A.16: is_all_or_one.m: A validation function, that checks if the input equals “all”. or “one”. It is used by “match_gcps.m” to determine if all, or just one of the GCP candidates are to be used to create topological point patterns (TPPs).

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_all_or_one( input_arg )
%% Discription
```

```
% IS_ALL_OR_ONE checks if the input is equal to the strings 'all' or 'one
    '.
% The case does not matter.

%% Check

output_arg = strcmpi(input_arg, 'all') || strcmpi(input_arg, 'one')';

end
```

Script A.17: is_binimg.m: A validation function that checks if the given input is a valid binary image.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%             Copyright (c) 2016 Sindre Nistad
function output_arg = is_binimg( input_arg )
%% Discription
% A function that checks is the given input matches the criteria of a
% binary image, i.e. a two dimentinal matrix that consists only of logical
% enteries.
%

%% Check
output_arg = islogical(input_arg) && ...
             numel(size(input_arg)) == 2;

end
```

Script A.18: is_boundary.m: A validation function, that checks if the boundaries are inclusive or exclusive.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%             Copyright (c) 2016 Sindre Nistad
function output_arg = is_boundary( input_arg )
%% Discription
% Checks is the given input is a valid mode for a boundary condition;
% 'Inclusice' or 'Exclusive'.

%% Check
output_arg = strcmp(input_arg, 'Inclusive') || ...
             strcmp(input_arg, 'Exclusive');

end
```

Script A.19: is_candidate_point_lists.m: A validation function, that checks that the input is a cell-array of candidate matchings (CMs).

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%             Copyright (c) 2016 Sindre Nistad
function output_arg = is_candidate_point_lists( input_arg )
%% Discription
```

```
% IS_CANDIDATE_POINT_LISTS checks is the given input is a cell array of candidate
% points. That is, a cell array whose elements are matrices of size n x 4
% or n x 6 of numerical data.

%% Check
output_arg = iscell(input_arg);
dims = size(input_arg{1}, 2);
if mod(dims, 2) ~= 0
    output_arg = false;
    return
end

for i = 1:numel(input_arg)
    if size(input_arg{i}, 2) ~= dims
        output_arg = false;
        return
    end
    A = input_arg{i};
    AA = A(:, 1:dims / 2);
    BB = A(:, dims / 2 + 1 : dims);
    output_arg = output_arg && is_point_list(AA) && is_point_list(BB);
end


end
```

Script A.20: is_coordinate_system.m: A validation function, that checks whether the given coordinate system is valid.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_coordinate_system( input_arg )
%% Discription
% IS_COORDINATE_SYSTEM checks if the coordinate system of a point set is
% valid.

%% Check
output_arg = strcmpi(input_arg, 'xy') || ...     % cartesian x-y
             strcmpi(input_arg, 'yx') || ...     % cartesian y-x
             strcmpi(input_arg, 'ne') || ...     % Projection Northing-Easting
             strcmpi(input_arg, 'en');           % Projection Easting-Northing

end
```

Script A.21: is_custom.m: A validation function that checks if the input is a function that can be used to constrain the morphology of a GCP candidate.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_custom( input_arg )
```

```
%% Discripption
% Checks if the given input is a valid custom input for
% "prune_morphology.m". It must contain which properties it should
% extract from regionprops, a function to act on these properties, and an
% interval for which the output from the function should be within in
    order
% to be considered OK. If the field 'Outside' is set to true, the oposite
% is true.

%% Check structure
if ~isstruct(input_arg)
    output_arg = false;
    return
end

if isstruct(input_arg) && isempty(input_arg)
    output_arg = true;
    return
end

output_arg = any(strcmp('Interval', fieldnames(input_arg))) && ...
             any(strcmp('Properties', fieldnames(input_arg))) && ...
             any(strcmp('Function', fieldnames(input_arg))) && ...
             any(strcmp('NecessaryProperties', fieldnames(input_arg))) &&
    ...
             any(strcmp('Mode', fieldnames(input_arg)));
if ~output_arg
    return
end


%% Check content
if any(strcmp('Outside', fieldnames(input_arg)))
    output_arg = islogical(input_arg.Outside);
    return
end

output_arg = is_interval(input_arg.Interval) && ...
             is_properties(input_arg.Properties) && ...
             is_function(input_arg.Function) && ...
             is_properties(input_arg.NecessaryProperties) && ...
             is_valid_mode(input_arg.Mode);

end
```

Script A.22: is_dem_or_disabled.m: A validation function that checks whether the input can be considered a digital elevation model (DEM).

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_dem_or_disabled( input_arg )
%% Discription
% IS_DEM_OR_DISABLED checks if the input argument is a digital elivation
% model, or disabled, i.e. false.

```

```
%% Check
output_arg = (islogical(input_arg) && input_arg == false) || ...
             ischar(input_arg) || ismatrix(input_arg);
end
```

Script A.23: is_fraction.m: A validation function that checks if the input is scalar (single number) between 0 and 1 inclusive.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_fraction( input_arg )
%% Discription
% IS_FRACTION chekcs that the given argument is a real number between 0
    and
% 1.

%% Check
output_arg = is_number(input_arg) && 0 <= input_arg && input_arg <= 1;

end
```

Script A.24: is_function.m: A validation function that checks whether the input is a function handle in MATLAB.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_function( input_arg )
%% Discription
% Checks is the given input is a function handle.

%% Check
output_arg = isa(input_arg, 'function_handle');

end
```

Script A.25: is_image: A validation function that checks if the input is a valid image, or an $n \times m \times 3$ matrix.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_argument = is_image( input_argument )
%% Discription
% Checks that the given input is a 3 banded image.

%% Initializing

img_size = size(input_argument);
dims = numel(img_size);

%% Check

```

```
output_argument = dims == 3 && (img_size(3) == 3 || img_size(3) == 4);  % There might be an alpha chanel

end
```

Script A.26: is_image_or_path.m: A validation function that checks whether the input is an image, or a path to an image.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_image_or_path( input_arg )
%% Discription
% IS_IMAGE_OR_PATH checks if the input is a valid image or that it is a
% path to an image.

%% check
output_arg = ischar(input_arg) || is_image(input_arg);

end
```

Script A.27: is_images: A validation function that tests whether the input is a valid cell-array of images, as defined in Script A.25.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_images( input_arg )
%% Discription
% IS_IMAGES checks that the given input is a cell array of 3 banded images.

%% Initializing

num_images = numel(input_arg);

%% Check
output_arg = iscell(input_arg);
for i = 1:num_images
    output_arg = output_arg && is_image(input_arg{i});
end

end
```

Script A.28: is_integer.m: A validation function, that tests whether the input is a scalar integer. This works with any number type (e.g. double, float, int, uint8), which the built-in function does not.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_integer( input_arg )
%% Discription
% IS_INTEGER checks that the given input is a positive scalar integer.

```

```
%% Check
output_arg = mod(input_arg, 1) == 0 && all(size(input_arg) == [1 1]);

end
```

Script A.29: is_interval.m: A validation function that tests whether the input is an interval such as $[2, 7]$.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_interval( input_arg )
%% Discription
% Checks that the given input is an interval, i.e. There are only two
% elements in the vector, and that the first element is smaller than the
% last element.

%% Check
output_arg = isvector(input_arg) && ...
             numel(input_arg) == 2 && ...
             input_arg(1) < input_arg(2);

end
```

Script A.30: is_interval_or_disabled: A validation function that tests whether the input is an interval, as defined in Script A.29, or disabled, i.e. set to `false`. If the input is `true`, however, the default value will be used.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_interval_or_disabled( input_arg )
%% Discription
% Checks that the given input is an interval, i.e. there are only two
% elements in the vector, and that the first element is smaller than the
% last element, or if it is disabled, i.e. false. If it is set to be true,
% it will later be assumed that the default values are to be used.

%% Check
output_arg = is_interval(input_arg) || islogical(input_arg);

end
```

Script A.31: is_min_max_std_mean.m: A validation function that tests whether the input is one of the modes “min-max” or “std-mean”. The first says the mode is to use the minimum and maximum value as limits In some cases it also indicates the use of quantiles. The latter specifies a mode using the mean value and the standard deviation as the interval to be used. In some cases, the standard deviation van be given a weight, so that the interval is extended or contracted.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_min_max_std_mean( input_arg )
```

```
%% Discription
% Checks is the given input is a 'min-max', 'max-min', 'std-mean', or
% 'mean-std'; valid modes.

%% Check
output_arg = strcmpi(input_arg, 'min-max') || ...
             strcmpi(input_arg, 'std-mean') || ...
             strcmpi(input_arg, 'max-min') || ...
             strcmpi(input_arg, 'mean-std');

end
```

Script A.32: is_number.m: A validation function that tests whether the input is a single number, i.e. a scalar.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_number( input_arg )
%% Discription
% IS_NUMBER checks if the given input is a single number, or a vector of
% size [1 1].

%% Check
output_arg = isnumeric(input_arg) && all(size(input_arg) == [1 1]);

end
```

Script A.33: is_number_or_disabled.m: A validation function that tests whether the input is a single number (i.e. scalar), or disabled, (i.e. `true`, in which case a default value is used, or `true`).

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_number_or_disabled( input_arg )
%% Discription
% IS_NUMBER_OR_DISABLED checks if the input argument is a single number,
    or
% that it is logical. For later use, if the input argument is set to ture,
% it will be assumed that the default values are to be used, whereas if it
% is false, it will be assumed that it is disabled.

%% Check
output_arg = (isnumeric(input_arg) && all(size(input_arg) == [1 1])) ||
    ...
             islogical(input_arg);

end
```

Script A.34: is_point_list.m: A validation function that tests whether the input is a valid list of points, e.i. an $n \times 3$, or $n \times 2$ matrix.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
```

```
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_point_list( input_arg )
%% Discription
% IS_POINT_LIST checks is the given input is an n x 2 or n x 3 matrix of
% numerical data.

%% Check
output_arg = isnumeric(input_arg) && (size(input_arg, 2) == 2  || size(
    input_arg, 2) == 3);

end
```

Script A.35: is_point_list_or_path.m: A validation function that tests whether the input is a valid list of points, as defined in Script A.34. The input can also be a path to such a list.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_point_list_or_path( input_arg )
%% Discription
% IS_POINT_LIST_OR_PATH checks is the given input is an n x 2 or n x 3
% matrix of numerical data, or that is a path to such a list.

%% Check
output_arg = is_point_list(input_arg) || ischar(input_arg);

end
```

Script A.36: is_positive_integer: A validation function that tests whether the input is a positive integer, as defined in Script A.28.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_positive_integer( input_arg )
%% Discription
% IS_POSITIVE_INTEGER checks that the given input is a positive scalar
% integer.

%% Check
output_arg = is_integer(input_arg) && input_arg > 0;

end
```

Script A.37: is_positive_number.m: A validation function that tests whether the given input is a positive number, as defined in Script A.32.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_positive_number( input_arg )
%% Discription
% IS_POSITIVE_NUMBER checks if the given input is a single number, or a
% vector of size [1 1], and that is greater or equal to 0.
```

```
9
10 %% Check
11 output_arg = is_number(input_arg) && input_arg >= 0;
12
13 end
```

Script A.38: is_properties.m: a validation function that tests whether the given input matches one or more of the legal parameters for the MATLAB function regionprops.

```
1  % This Source Code Form is subject to the terms of the Mozilla Public
2  % License, v. 2.0. If a copy of the MPL was not distributed with this
3  % file, You can obtain one at http://mozilla.org/MPL/2.0/.
4  %            Copyright (c) 2016 Sindre Nistad
5  function output_arg = is_properties( input_arg )
6  %% Discription
7  % IS_PROPERTIES checks whether or not the given input can be considered a
8  % list of valid properties to be used in regionprops. It does this by
9  % checking that the input is either a cell array of string, or a single
10 % string.
11 %
12
13
14 %% List of valid strings:
15 properties = {'all', 'basic', 'Area', 'Centroid', 'BoundingBox', ...
16     'SubarrayIdx', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
       ...
17     'Orientation', 'ConvexHull', 'ConvexImage', 'ConvexArea', 'Image', ...
18     'FilledImage', 'FilledArea', 'EulerNumber', 'Extrema', ...
19     'EquivDiameter', 'Solidity', 'Extent', 'PixelIdxList', 'PixelList',
       ...
20     'Perimeter', 'PerimeterOld', 'PixelValues', 'WeightedCentroid', ...
21     'MeanIntensity', 'MinIntensity', 'MaxIntensity', ''};
22
23 %% Check
24
25 if iscellstr(input_arg)
26     output_arg = true;
27     for i = 1:numel(input_arg)
28         output_arg = output_arg && ...
29             any(strcmp(input_arg(i), properties));
30     end
31 elseif ischar(input_arg)
32     output_arg = any(strcmp(input_arg, properties));
33 else
34     output_arg = false;
35 end
36
37
38 end
```

Script A.39: is_replace_mode.m: A validation function that tests whether the given input is a valid mode for what to do with GCP candidates that are too close to each other.

```
1 % This Source Code Form is subject to the terms of the Mozilla Public
2 % License, v. 2.0. If a copy of the MPL was not distributed with this
3 % file, You can obtain one at http://mozilla.org/MPL/2.0/.
4 %            Copyright (c) 2016 Sindre Nistad
```

```
function output_arg = is_replace_mode( input_arg )
%% Discription
% Checks is the given input is a valid mode for what to do with points
    that
% are too close to eachother.

%% Check
output_arg = strcmpi(input_arg, 'probable') || ...
             strcmpi(input_arg, 'average')  || ...
             strcmpi(input_arg, 'remove');

end
```

Script A.40: is_sample_data_or_disabled.m: A validation function that tests whether the given input is an $n \times 3$ matrix, a path to it, or disabled. If it is disabled, a default value will be used instead.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_sample_data_or_disabled( input_arg )
%% Discription
% IS_SAMPLE-DATA_OR_DISABLED checks that the input data is a valid sample
% of RGB vales, or a path, or disabled.

%% Check
output_arg = (isnumeric(input_arg) && size(input_arg, 2) == 3) || ...
              ischar(input_arg) || ...
              (islogical(input_arg) && input_arg == false);

end
```

Script A.41: is_structure_element.m: A validation function that tests whether the given input is a structure element in the form of a "strel" class, or a template (i.e. a binary image).

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_structure_element( input_arg )
%% Discription
% IS_STRUCTURE_ELEMENT checks if the input argument is a structure element
    .
% It can either be a strel or a binary image.

%% Check
output_arg = isa(input_arg, 'strel') || is_binimg(input_arg);

end
```

Script A.42: is_valid_mode.m: A validation function that tests whether the given input is a valid mode for Script A.55.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
```

```
function output_arg = is_valid_mode( input_arg )
%% Discription
% IS_VALID_MODE checks that the given (set of) string(s) is a valid mode
% for the function "prune_morphology.m".

%% Define valid modes
modes = {'Interval', 'Function', 'IntervalFunction'};

%% Check
output_arg = any(strcmp(input_arg, modes));

end
```

Script A.43: is_valid_orientation_algorithm.m: A validation function that tests whether the given input is a valid algorithm for finding the absolute orientation parameters.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = is_valid_orientation_algorithm( input_arg )
%% Discription
% IS_VALID_ORIENTATION_ALGORITHM cheks that the input is a valid choise of
% algorithm for the absolute positioning problem.

%% Check
output_arg = strcmpi(input_arg, 'ShinjiUmeyama') || ...
             strcmpi(input_arg, 'Horn') || ...
             strcmpi(input_arg, 'HornHilden') ;
end
```

Script A.44: limits.m: Creates an interval of limits from a given dataset by using quantiles, or the mean and a multiple of the standard deviation. All the limits are per column, or band, of the sample data.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function res = limits(values, mode, opt)
%% Discrition
% This is a function that gathers the limits_min_max and limits_mean_std
% functions into one umbrella function.
% The values are a collection of values (a matrix) where each column
% represents a sepearate band. The mode specifies how the limits are to be
% computed: 'min-max' or 'mean-std'. The opt argument specifies any
% auxillary information that is requiered. For min-max, it is the
% percentile of quantiles to be used. 0 or 1 gives min/max, while for
% 'mean-std' the option is the weight to be given to the standard
% deviation.

%% Initializing
default = 'min-max';

if nargin < 2
    mode = default;
elseif nargin < 3
```

```
    if strcmp(mode, 'min-max')
        opt = 0;
    elseif strcmp(mode, 'mean-std')
        opt = 1;
    else
        warning(strcat('Invalid mode, using default, (', default, ').'));
    end
end

%% Getting limits

if strcmp(mode, 'min-max')
    res = limits_min_max(values, opt);
elseif strcmp(mode, 'mean-std')
    means = mean(values);
    stds = std(values);
    res = limits_mean_std(means, stds, opt);
else
    res = limits(values, default);
end

end
```

Script A.45: limits_mean_std.m: A helper function for Script A.44 that creates an interval of size $\pm a\sigma$ around the mean of the sample data. $a \in \mathbb{R}$, and σ is the standard deviation of the data. All the limits are per column, or band, of the sample data.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function res = limits_mean_std(means, stds, weights)
%% Description
% A function that creates minimum and maximum values for each of the bands
    .
% It gives a matrix with minimum and maximum values. The paramters is
% means, which is a vector of means for each band, while stds is a vector
% for the standard deviation of each band, while weights is a scalar or
% vector of weights for the standard deviation.
% The result will be in the form
%     mean(1) - wheights(1) * stds(1), means(1) + wheights(1) * stds(1)
%     mean(2) - wheights(2) * stds(2), means(2) + wheights(2) * stds(2)
%                                    .
%                                    .
%                                    .

%% Checking consistency
if ~(isvector(means) && isvector(stds) && ...
        (isvector(weights) || isscalar(weights)))
    error('The means, standard deviations, and the weights are not vecotrs
    , or weights is not a scalar');
elseif numel(means) ~= numel(stds)
    error('The number of means and standard deviations is not the same');
elseif numel(means) ~= numel(weights) && numel(weights) ~= 1
    error('The number of weights is not the same as the means and standard
     deviations, and weights is not a scalar');
end
```

```

%% Initializing
res = zeros(numel(means), 2);
if isscalar(weights)
    temp = zeros(size(means));
    temp(:) = weights;
    weights = temp;
end

for i = 1:numel(means)
    variance = weights(i) * stds(i);
    res(i,:) = [means(i) - variance, means(i) + variance];
end
```

Script A.46: limits_min_max.m: A helper function to Script A.44 that creates an interval. The interval can either be the minimum and maximum of the sample data, or it can be an arbitrary quantile of the data. All the limits are per column, or band, of the sample data.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function res = limits_min_max(vals, p)
%% Discription
% Creates a matrix of minimum and maximum values if p is not given. If p
    is
% between 0 - 1, then the function will return the p and 1 - p quantiles
    of
% the data for each band.

%% Initializatinon
if ~exist('p', 'var')
    p = 0;
elseif p > 1
    p = 1;
elseif p < 0
    p = 0;
end

size_vals = size(vals);
num_bands = size_vals(2);
res = zeros(num_bands, 2);

%% Getting values
for i = 1:num_bands
    % A little MATLAB hack to get the values into a 1 x 2 vector.
    q = quantile(vals(:,i), [p, 1 - p]);
    res(i, :) = [q(1), q(2)];
end
```

Script A.47: load_geojson.m: A function that loads the content of a GeoJSON file and returns a list of points, the coordinate system used and the names of the points.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
```

```
function [points, crc, names] = load_geojson(path)
%% Discription
% Loads the geojson object located at *path*. It returns a matrix of
% points; each row is a point of the form [x y z] or [E N H] for a propper
% projection. The *crc* output is the reference coordinate system used in
% the given file.
% This function is dependent on JSONlab.

%% Load JSON object
json = loadjson(path);

num_points = numel(json.features);
points = zeros(num_points, 3);
names = cell(num_points, 1);

crc = json.crs.properties.name;

%% Fill the matriex
for i = 1:num_points
    point = json.features{i};
    coordinates = point.geometry.coordinates;
    name = point.properties.name;
    points(i, :) = coordinates;
    names{i} = name;
end

end
```

Script A.48: mahal_dist.m: A customized version of the built-in function `mahal` to effectively calculate the Mahalanobis distance of an entire image.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function res = mahal_dist(Y, X, m)
%% Disctiption
% MAHAL_DIST runs the bulit in function mahal for an image, instead of
    just
% an array of points. X is the reference sample, while Y is the points to
% whish we wish to get the distances.
% If the input Y is an image, the output will be an image of distances.
% This should hopefully be faster on larger data than using the
    mahalanobis
% distance per element.

%% Creating the function
img_size = size(Y);
if img_size(2) == 3 && size(img_size, 2) == 2
    res = mahal(Y, X);
elseif nargin == 2
    res = mahal(reshape(Y, [prod(img_size(1:2)), 3]), X);
    res = reshape(res, [img_size(1), img_size(2)]);
else
    % It is assumed that X is the invers of the covariance matrix
    % and that m is the mean
    val = reshape(Y, [img_size(1) * img_size(2), img_size(3)]);
```

```
    res = sum((bsxfun(@minus, val, m) * X) .* bsxfun(@minus, val, m),2);
    res = reshape(res, [img_size(1), img_size(2)]);
end

end
```

Script A.49: make_outside_interval_checker.m: Creates a function that checks if a given value is inside, or outside a specified interval. The boundaries can be chosen to be inclusive, or exclusive independently.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function outside = make_outside_interval_checker(min_boundary,
    max_boundary)
%% Discription
% MAKE_OUTSIDE_INTERVAL_CHECKER creates a function that checjs if a value
% is inside an interval with inclusive, or exclusive bounderies.

%% Initilaizing
if strcmp(min_boundary, 'Exclusive')
    outside_min = @(value, minimum) value <= minimum;
elseif strcmp(min_boundary, 'Inclusive')
    outside_min = @(value, minimum) value < minimum;
else
    error('Not a valid maximum boundary. Use ''Exclusive'' or ''Inclusive'
    '');
end

if strcmp(max_boundary, 'Exclusive')
    outside_max = @(value, maximum) value >= maximum;
elseif strcmp(max_boundary, 'Inclusive')
    outside_max = @(value, maximum) value > maximum;
else
    error('Not a valid maximum boundary. Use ''Exclusive'' or ''Inclusive'
    '');
end

%% Create function
outside = @(value, minimum, maximum) outside_min(value, minimum) || ...
                                     outside_max(value, maximum);
end
```

Script A.50: match_gcps.m: An implementation of the TPP algorithm for finding the correspondence between two sets of points. Adapted from Li and Briggs (2006).

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function varargout = match_gcps(image_points, gcps, varargin)
%% Description
% This function matches ground controll points from a orthophoto with the
% actual coordinates ground controll points. The input points is the
% relative coordinates of the ground controll points in the orthophoto /
% image. Not all of these points needs to be actual coordinates. The
```

```
11 % function is able to handle missing points, and false positives.
12 % The Input prameter gcps is a list of the world coordinates of the ground
13 % controll points.
14 %
15 % USAGE:
16 % matches = match_gcps(image_points, gcps)
17 % matches = match_gcps(image_points, gcps, options)
18 % [CM, ST, RMSE] = match_gcps(image_points, gcps, 'GetOptimal', true, ___)
19 %
20 %
21 %
22 % Valid options
23 %   'ImageTPPMode'      The mode of how the topological point pattern is
       to
24 %                       be used. The valid modes area 'all', 'one', If
25 %                       'one' is selected, a single point will be chosen
       as
26 %                       an anchor point. Unless 'ImageTPPIndex' is given
       as
27 %                       well, this point will be chosen at random. If 'all
       '
28 %                       is given insted, the algorithm will choose the
29 %                       point pattern for the image that gives the best
30 %                       RMSE with all the possible point patterns in the
31 %                       ground controll point set. This is slower than
32 %                       selecting 'one'.
33 %                       Default is 'one'.
34 %   'ImageTPPIndex'     The index of which point to be used as the anchor
35 %                       point in the set of points from the image. If the
36 %                       index is negative or beyond the number of points
       in
37 %                       the set, a point will be chosen at random.
38 %                       Default is -1 (random).
39 %   'MinimumMatches'    The minimum points necessary for a set of matching
40 %                       points to be considered for a true matching.
41 %                       Default is the number of unknowns in the absolute
42 %                       matching problem plus one, e.g. is the points are
43 %                       in 2D, this value will be 2, while if the points
44 %                       are in 3D, it will be 3.
45 %   'RadiusThreshold'   The threshold used when comparing radiuses. If the
46 %                       difference in radia between the point pattern in
47 %                       the image and the point pattern in the ground
48 %                       control points is less than the given threshold,
       it
49 %                       will be considered a match.
50 %                       Default is 0.1.
51 %   'AngleThreshold'    The threshold used when comparing angles. If the
52 %                       difference in angles between the point pattern in
53 %                       the image and the point pattern in the ground
54 %                       control points is less than the given threshold,
       it
55 %                       will be considered a match.
56 %                       Default is 0.1.
57 %   'DegreesFreedom'    The number of points that is desireable beyond
       what
58 %                       is necessary to get a unique solution to the
59 %                       absolute orientation problem. Default is 1.
```

```
%   'MinMatchedPoints'  The minimum number of points that are to matched.
%                       Default is the number of points to get a uniqe
%                       solution.
%   'GetIndices'        A boolean falg toggeling wether or not just the
%                       indices of matching points are to be returned. If
%                       true, only the indices of the points that are
%                       matching eachother will be returned. If false, the
%                       full coordinates will be returned in an n x 2 * d
%                       matrix where d is the dimentionality of the points
    .
%                       Default is false.
%   'GetOptimal'        A boolean flag that toggles wether the output is
%                       the best candidate matching, along with the
%                       transformation parameters and the root mean square
%                       error of the transformation.
%                       Default is false.
%                       NB: When using this option, 'GetIndices' will not
%                       be a valid argument, and will be ignored.
%
%   'OrientationAlgorithm'  The algorithm to be used when computing the
%                       optimal absolute orientation. This option is anly
%                       applicable when the 'GetOptimal'-flag is set to
%                       true. The valid options are 'ShinjiUmeyama' and
%                       'Horn'.
%                       Default is 'Horn'.
%
%   'MaxDistance'       Specifies the maximum allowed scaled distance
%                       between the different points in the pattern.
%                       Default is Inf, i.e. no limit.
%
%   'ImageLimit'        Toggles whether the set of topological points
%                       from the image are limited to the maximum
%                       distance of the topological points of the
%                       ground control points.
%                       Default is false.
%
%   'GCPLimit'          Toggles whether the set of topological points
%                       from the ground control points are limited to
%                       the maximum distance of the topological
%                       points of the image.
%                       Default is false.
%
%   'UseHornScale'      Toggles whether the scale factor suggested in
%                       Horn (1986), and Horn et. al. (1988) is to be
%                       used instead of the scale factor suggested in
%                       Umeyama (1991) for the algorithm
%                       'ShinjiUmeyama'.
%                       Default is false.
%
% If 'GetIndices' is set to true matches will be a cell array of
    cellarrays
% of matrices having rows [i, j] where the indices is the points in of
% *image_points* and *gcps* respectively. If it is set to false, however,
% it will return a cell array each containing a n x 2 * d matrix of the
% points in image_points and gcps. The points in the first is in the first
% two columns of the cell array, while the latter is in the latter two.
% I.e. [x_i y_i N_j E_j]. d is the number of dimention in the points.
```

```
%
% NB: If *image_points* and *gcps* have different dimentions, the lowest
% dimention will be used, i.e. id on is 2D and the other is 3D, the output
% will only be 2D.

%% Parse input

i_p = inputParser;
i_p.FunctionName = 'MATCH_GCPS';

% Requiered
i_p.addRequired('image_points', @is_point_list);
i_p.addRequired('gcps', @is_point_list);

% Optional: What to remove
i_p.addParameter('ImageTPPMode', 'one', @is_all_or_one);
i_p.addParameter('ImageTPPIndex', -1, @is_number);
i_p.addParameter('MinimumMatches', -1, @is_number);                    %
    Default is a 5 if the points are 2D, and 7 if they are 3D
i_p.addParameter('RadiusThreshold', 0.21, @is_number);
i_p.addParameter('AngleThreshold', 0.21, @is_number);
i_p.addParameter('DegreesFreedom', 1, @is_positive_integer);
i_p.addParameter('GetIndices', false, @islogical);
i_p.addParameter('GetOptimal', false, @islogical);
i_p.addParameter('ImageCoordinateSystem', 'xy', @is_coordinate_system);
i_p.addParameter('GCPCoordinateSystem', 'NE', @is_coordinate_system);
i_p.addParameter('OrientationAlgorithm', 'Horn',
    @is_valid_orientation_algorithm);
i_p.addParameter('MaxDistance', Inf, @is_positive_number);
i_p.addParameter('GCPLimit', false, @islogical);
i_p.addParameter('ImageLimit', false, @islogical);
i_p.addParameter('UseHornScale', false, @islogical);


i_p.addParameter('Debug', false, @islogical);
i_p.addParameter('DebugPath', 'D:\Users\sindr\Dropbox\Dokumenter\Skole\
    NTNU\Master\Masteroppgave\data\resultater\kandidater\', @ischar);


i_p.parse(image_points, gcps, varargin{:});

%% Deal with the input
input = i_p.Results;

% Required
image_points            = input.image_points;
gcps                    = input.gcps;

% Optional
i                       = input.ImageTPPIndex;
mode                    = input.ImageTPPMode;
image_coor              = input.ImageCoordinateSystem;
gcp_coor                = input.GCPCoordinateSystem;

sufficient_points       = input.MinimumMatches;

delta_r                 = input.RadiusThreshold;
```

```
delta_theta             = input.AngleThreshold;

dof                     = input.DegreesFreedom;

use_indices             = input.GetIndices;
use_verification        = input.GetOptimal;

max_distance            = input.MaxDistance;
image_limit             = input.ImageLimit;
gcp_limit               = input.GCPLimit;
use_horn_scale          = input.UseHornScale;

% Debug
debug_mode              = input.Debug;
debug_path              = input.DebugPath;

orientation_algorithm   = input.OrientationAlgorithm;

%% Initialization
dim = min([size(image_points, 2), size(gcps, 2)]);

% Determining the minimum needed points to get unique solution
minimum_num_points = ceil(((dim - 1) + dim + 1) / dim); % Angles,
    translation and scaling, and there are *dim' number of equations per
    point
if sufficient_points <= 0
    sufficient_points = (minimum_num_points * dim + dof) / dim;
elseif sufficient_points <= minimum_num_points
    warning('The number of minimum matches is lower or equal to the number
     of points requiered to solve the absolute orientation problem');
end

if use_verification
    % We need the actual points.
    use_indices = false;
end

if strcmpi(image_coor, 'yx')
    image_points = [image_points(:, 2) image_points(:, 1)];
end
if strcmpi(gcp_coor, 'NE')
%     gcps = [gcps(:, 2) gcps(:, 1)];
end

image_points = image_points(:, 1:dim);
gcps = gcps(:, 1:dim);


if use_verification
    varargout = cell(3, 1);
else
    varargout = cell(1);
end

if gcp_limit && image_limit
    TPPs_image = create_TPPs_for_image(image_points, i, max_distance, mode
    );
```

```
222     TPPs_gcp = create_TPPs_for_gcp(gcps, max_distance);
223     d = min([find_maximum_distance(TPPs_image), ...
224              find_maximum_distance(TPPs_gcp), max_distance]);
225     TPPs_image = prune_tpp(TPPs_image, d);
226     TPPs_gcp = prune_tpp(TPPs_gcp, d);
227 elseif gcp_limit
228     % The GCPs are limited by the maximum distance of image TPP
229     TPPs_image = create_TPPs_for_image(image_points, i, max_distance, mode
        );
230     d = find_maximum_distance(TPPs_image);
231     TPPs_gcp = create_TPPs_for_gcp(gcps, min([max_distance, d]));
232 elseif image_limit
233     TPPs_gcp = create_TPPs_for_gcp(gcps, max_distance);
234     d = find_maximum_distance(TPPs_gcp);
235     TPPs_image = create_TPPs_for_image(image_points, i, min([max_distance,
         d]), mode);
236 else
237     TPPs_gcp = create_TPPs_for_gcp(gcps, max_distance);
238     TPPs_image = create_TPPs_for_image(image_points, i, max_distance, mode
        );
239 end
240
241 %% Matching
242
243 matches = match_points(TPPs_image, TPPs_gcp, sufficient_points, delta_r,
        delta_theta, mode, debug_mode, debug_path);
244
245 %% Finallizing the output
246
247 n_matches = numel(matches);
248
249 for ii = 1:n_matches
250     CM = matches{ii};
251     acm = CM{1};
252     idx_gcp = CM{2}; idx_img = CM{3};
253     matches{ii} = [TPPs_image{idx_img}.Indices(acm(:, 1)), TPPs_gcp{
        idx_gcp}.Indices(acm(:, 2))];
254 end
255
256 if ~use_indices
257     for ii = 1:n_matches
258         CM = matches{ii};
259         matches{ii} = [image_points(CM(:, 1), :), gcps(CM(:, 2), :)];
260     end
261 end
262 if use_verification
263     [CM, ST, RMSE] = verification_algorithm(matches, orientation_algorithm
        , use_horn_scale);
264     varargout{1} = CM; varargout{2} = ST; varargout{3} = RMSE;
265 else
266     varargout{1} = matches;
267 end
268
269
270 end
271
272 %% TOPOLOGICAL_POINT_PATTERN
```

```
%=========================================================================
function TPP = topologival_point_pattern( points, i, d )
%% Discription
% TOPOLOGICAL_POINT_PATTERN computes the topological point pattern (TPP) of
% the given set of points, using the i-th point as an anchor point.
% The output is a struct with the fields 'TPP', 'Indices', 'AnchorPoint',
% 'AnchorPointIndex', and 'ScalingFactor'.
% More specifically:
%   'TPP' is a sorted list of polar cooredinates for each
% point in the input set. The set is sorted lexicographically (r, theta).
%   'Indices' is a set of indices for a one-to-one correspondence with the
% points in the given set. They are indexed by their order in TPP. I.g. if
% the i-th element of I is j, that means that the i-th element in *TPP* is
% equivelent to the j-th point in *points*.
%   'AnchorPoint' is the absolute position of the anchor point.
%   'AnchorPointIndex is the index that was used when computing the
% topological point pattern.
%   'ScalingFactor' is the unit distance of r, i.e. the distance between the
% anchor point and its closest neighbor.

%% Initialization
n = size(points, 1) - 1;

anchor_point = points(i, :);
if nargin == 2
    d = Inf;
end

%% Procedure
% Translate all points relative the anchor point
points = bsxfun(@minus, anchor_point, points);

% Calculate distances
D = sqrt(sum(points.^2, 2));

% Find the closest point to the anchor point
D(D == 0) = Inf;    % Hack to avoid getting the same point when using min(D(D ~= 0)) or min(nonzero(D))
[~, I] = min(D);
D(D == Inf) = 0;    % Hack to avoid getting the same point when using min(D(D ~= 0)) or min(nonzero(D))

% Since the points have been translated, the vector of the prixipal axis
% coincides with the closest point, when we treat it as a vector
principal_axis = points(I, :);
principal_angle =  atan2(principal_axis(2), principal_axis(1));

scaling = sqrt(principal_axis(1)^2 + principal_axis(2)^2);

r = D / scaling;
theta = rem(atan2(points(:, 2), points(:, 1)) - principal_angle, pi);

% Remove those that are too far away
I = r <= d;
r = r(I);
```

```
theta = theta(I);

% Make the angle interval [0 2pi] instead of [-pi, pi]
theta(theta < 0) = theta(theta < 0) + 2*pi;

%% Computing outputs
% Topoligical point pattern
[~, I] = sortrows([r min([theta, 2 * pi - theta], [], 2)]);
pattern = [r(I), theta(I)];
pattern(1, 2) = 0;  % Ensure that the angle of the anchor point is zero.

% Anchor point
ap = anchor_point;

% Scaling factor
s = scaling;

% Gather it all together in a struct
TPP = struct(...
    'TPP', pattern, ...
    'Indices', I, ...
    'AnchorPoint', ap, ...
    'AnchorPointIndex', i, ...
    'ScalingFactor', s);

end

%% IS_SUBSET
%=========================================================================
function res = is_subset(TPPa, TTPb)
%% Discription
% Checks if TPPa is a subset of TTPb

%%
res = true;

end

%% PRUNE_TPPs
%=========================================================================
function TPP = prune_tpp(TPP, max_distance)
%% Discription
% Removes all points that are further away from the anchor point than
% the given distance.

%% Prune
for i = 1:numel(TPP)
    tpp = TPP{i};
    pattern = tpp.TPP;
    I = pattern(:, 1) <= max_distance;
    tpp.TPP = pattern(I, :);
    tpp.Indices = tpp.Indices(I);
    TPP{i} = tpp;
end

end

```

```
383
384 %% EXTRACT_MATCHING_PAIRS
385 %=========================================================================
386 function matches = extract_matching_pairs(TPPa, TPPb, delta_r, delta_theta
        )
387 %% Discription
388 % Extract the points that matches in the two given sets
389 % Input: Two sets of Topological Point Patterns; TPPa and TPPb, along with
390 % a set of thresholds; $\Delta{} r$ and $\Delta{} \theta$ for the minimal
391 % accepted difference between the radius and angle in polar coordinates
392 % respectively.
393
394 %% Initialization
395 n_a = size(TPPa, 1);
396 n_b = size(TPPb, 1);
397
398 matches = zeros(max([n_a, n_b]), 2);
399
400 i = 1;
401 j = 1;
402
403 %% Look for matching pairs
404
405 while i <= n_a && j <= n_b
406
407     theta_a = TPPa(i, 2);
408     theta_b = TPPb(j, 2);
409
410     diff_r     = abs(TPPa(i, 1) - TPPb(j, 1));
411     diff_theta = min([abs(theta_a - theta_b), ...
412                       theta_a + (2*pi - theta_b), ...% b is bellow the
        axis
413                       theta_b + (2*pi - theta_a)]); % a is bellow the axis
414     if diff_r <= delta_r && diff_theta <= delta_theta
415         matches(i, :) = [i j];
416         i = i + 1;
417         j = j + 1;
418     elseif TPPa(i, 1) > TPPb(j, 1)
419         j = j + 1;
420     elseif TPPa(i, 1) < TPPb(j, 1)
421         i = i + 1;
422     elseif i + 1 <= n_a && j + 1 <= n_b
423         % This means the points are close enough, but the angle is off
424         if abs(TPPa(i + 1, 1) - TPPb(j, 1)) <= diff_r && ...
425                 abs(TPPa(i, 1) - TPPb(j + 1, 1)) <= diff_r && ...
426                 abs(TPPa(i + 1, 2) - TPPb(j, 2)) < diff_theta && ...
427                 abs(TPPa(i, 2) - TPPb(j + 1, 2)) < diff_theta
428             % Both next candidates are closer than the previous, and both
429             % are within the theresholds of $r$ and $\theta$, so we choose
430             % the one with the smallest angle
431             if abs(TPPa(i + 1, 2) - TPPb(j, 2)) < abs(TPPa(i, 2) - TPP(j +
      1, 2))
432                 i = i + 1;
433             else
434                 j = j + 1;
435             end
436         elseif abs(TPPa(i + 1, 1) - TPPb(j, 1)) <= diff_r && abs(TPPa(i +
```

```
    1, 2) - TPPb(j, 2)) < diff_theta
            i = i + 1;
        elseif abs(TPPa(i, 1) - TPPb(j + 1, 1)) <= diff_r && abs(TPPa(i,
    2) - TPPb(j + 1, 2)) < diff_theta
            j = j + 1;
        else
            i = i + 1;
            j = j + 1;
        end
    else
        i = i + 1;
        j = j + 1;
    end
end

% Removes the empty rows
matches( ~any(matches, 2), :) = [];

end

%% MATCH_POINTS
%==========================================================================
function matches = match_points(TPPs_image, TPPs_gcp, sufficient_points,
    delta_r, delta_theta, mode, debug_mode, debug_path)

n_gcp = numel(TPPs_gcp);
n_img = numel(TPPs_image);

if strcmpi(mode, 'all')
    n = n_img;
else
    n = 1;
end

CCM = cell(n_gcp, n);

for ii = 1:n_gcp
    for jj = 1:n
        TPP_image   = TPPs_image{jj};
        TPP_gcp     = TPPs_gcp{ii};
        if is_subset(TPP_image, TPP_gcp)
            acm = extract_matching_pairs(TPP_image.TPP, TPP_gcp.TPP,
    delta_r, delta_theta);
            if debug_mode
                plot_TPPs(TPP_image.TPP, TPP_gcp.TPP);
                saveas(gcf, strcat(debug_path, num2str(ii), '-', num2str(
    jj)), 'png');
                close all
            end
            if size(acm, 1) >= sufficient_points
                CCM{ii, jj} = {acm, ii, jj};
            end
        end
    end
end

matches = remove_empty_cells(CCM);
```

```
489
490 if numel(matches) == 0 && sufficient_points >= 2
491     warning(strcat('The number of maching points is too much (', ...
492         num2str(sufficient_points), '). Trying again with', ' ', ...
493         num2str(sufficient_points - 1), ' matching points.'));
494     matches = match_points(TPPs_image, TPPs_gcp, sufficient_points - 1,
    delta_r, delta_theta, mode, debug_mode, debug_mode );
495 elseif numel(matches) == 0
496     error('There are no matching points.');
497 end
498
499 end
500
501 %% CREATE_TPPS_FOR_IMAGE
502 %=========================================================================
503 function TPPs_image = create_TPPs_for_image(image_points, i, max_distance,
     mode)
504 %% Discription
505 % Creates a collection of TPP from the image
506 n_img = size(image_points, 1);
507
508 if strcmpi(mode, 'one')
509     if i <= 0 || i > n_img
510         % Discrete Uniform from 1 to n_img inlusive
511         i = random('unid', n_img);
512     end
513     TPPs_image = {topological_point_pattern(image_points, i, max_distance)
    };
514 else
515     TPPs_image = cell(n_img, 1);
516     for ii = 1:n_img
517         TPPs_image{ii} = topological_point_pattern(image_points, ii,
    max_distance);
518     end
519 end
520
521 end
522
523 %% CREATE_TPPS_FOR_GCP
524 %=========================================================================
525 function TPPs_gcp = create_TPPs_for_gcp(gcps, max_distance)
526 %% Discription
527 % Creates a collection of topological point patterns from the points
528 % in the set of ground control points.
529
530 %% Initialization
531 n_gcp = size(gcps, 1);
532
533 TPPs_gcp = cell(n_gcp, 1);
534
535 %% Compute TPPs
536 for ii = 1:n_gcp
537     TPPs_gcp{ii} = topological_point_pattern(gcps, ii, max_distance);
538 end
539
540 end
541
```

```
%% FIND_MAXIMUM_DISTANCE
%==========================================================================
function d = find_maximum_distance(TPP)
%% Discription
% Finds the maximum distance in the given topological point pattern.

%% Find trhe distnace
d = 0;
for ii = 1:numel(TPP)
    if iscell(TPP)
        pattern = TPP{ii}.TPP;
    else
        pattern = TPP{ii};
    end

    tmp_d = pattern(end,1);
    if tmp_d > d
        d = tmp_d;
    end
end

end
```

Script A.51: mirror.m: Mirrors a set of points about a vertical, or horizontal line that goes through the center of the points.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function reflected_points = mirror(points, direction)
%% Discription
% REFLECT reflects all the points in the set *points* about a line trough
% the centre of the points. The direction of the line the points are to be
% reflected about can either be 'Horizontal', or 'Vertical'.

%%
means = mean(points);
r = bsxfun(@minus, points, means);

if strcmpi(direction, 'Horizontal')
    r = [r(:, 1) r(:,2) * -1];
elseif strcmpi(direction, 'Vertical')
    r = [r(:, 1) * -1 r(:,2)];
end
if size(points, 2) == 3
    r = [r, points(:, 3)];
end

reflected_points = bsxfun(@plus, r, means);

end
```

Script A.52: normalize.m: A utility function that normalizes a dataset linearly, so that the maximum has a value of 1, while the minimum gets a vale of 0.

```
% This Source Code Form is subject to the terms of the Mozilla Public
```

```
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function output_arg = normalize( input_arg )
%% Discription
% NORMALIZE normalizes the input in a linear fashion.

%% Normalize the data
output_arg = (input_arg - min(input_arg(:))) / (max(input_arg(:)) - min(
    input_arg(:)));

end
```

Script A.53: num_regions.m: A function that counts the number of regions in a binary image. Areas with a value of 1, are counted as foreground.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function num = num_regions( BW )
%% Discription
% NUM_REGIONS computes the number of seperated regions there are in the
% given binary image.

%% Compute number of regions
CC = bwconncomp(BW, 8);
num = CC.NumObjects;

end
```

Script A.54: plot_TPPs.m: A debugging, and inspection function that plots two sets of TPPs on top of each other, so that one can visually inspect if two sets should be a match. Useful when trying to find good parameters for the difference in radius and angle.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function plot_TPPs(TPPa, TPPb)
%% Discription
% PLOT_TPPS creates a scatter plot of the two topological point patterns,
% making it easy to compare them.

%% Plot

[x,y] = pol2cart(TPPa(:,2), TPPa(:,1));
scatter(x,y);
hold on
[x,y] = pol2cart(TPPb(:,2), TPPb(:,1));
scatter(x,y, 'x');


end
```

Script A.55: prune_morphology.m: A function that detects areas that does not conform to the specified limits for certain morphological features. An example would be areas that are too eccentric, or have the wrong area.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function right_morphology = prune_morphology(images, dist_fun,
    max_distance, varargin)
%% Discription

% Options:
% Eccentricity: Uses the option 'Eccesntricity' in regionprops. The
% pramameters is an interval [min, max]. If

%% Parse input

i_p = inputParser;
i_p.FunctionName = 'PRUNE_MORPHOLOGY';

% Requiered
i_p.addRequired('images', @is_image_or_images);       % Image
% Not required
% dist_fun requires max_distance
i_p.addOptional('dist_fun', @is_function);   % Distance function
% max_distance can be given without dist_fun
i_p.addOptional('max_distance', @isnumeric); % Max distance / threshold

% Optional: What to remove
i_p.addParameter('Area', false, @is_interval_or_disabled);
i_p.addParameter('Eccentricity', false, @is_interval_or_disabled);
i_p.addParameter('Solidity', false, @is_interval_or_disabled);
i_p.addParameter('AreaPerimeter', false, @is_interval_or_disabled);
i_p.addParameter('IgnoreArea', false, @is_interval_or_disabled);
i_p.addParameter('Tightness', false, @is_interval_or_disabled);   % Area /
     Bounding box
i_p.addParameter('Smoothing', false, @is_number_or_disabled);    %
    Gaussian smoothing with sigma equal to the given number
i_p.addParameter('Median', false, @islogical);
i_p.addParameter('NumElements', false, @is_number_or_disabled);
i_p.addParameter('Fill', false, @is_structure_element);
% Add
i_p.addParameter('Custom', struct([]), @is_custom);

i_p.parse(images, dist_fun, max_distance, varargin{:});

%% Dealing with the given input data
inputs = i_p.Results;

parameters = struct(...
    'Area',         create_parameter_structure(inputs.Area, {'Area'}, @(x)
     x, 'Interval'), ...
    'Eccentricity', create_parameter_structure(inputs.Eccentricity, {'
    Eccentricity'}, @(x) x, 'Interval'), ...
    'Solidity',     create_parameter_structure(inputs.Solidity, {'Solidity
    '}, @(x) x, 'Interval'), ...
    'AreaPerimeter',create_parameter_structure(inputs.AreaPerimeter, {'
```

```
    Area', 'Perimeter'},@(area, perimeter) area / perimeter, 'Interval'),
    ...
    'IgnoreArea',    create_parameter_structure(inputs.IgnoreArea, {'Area'
    }, @(x) x, 'Interval'), ...
    'Tightness',     create_parameter_structure(inputs.Tightness, {'Area',
    'BoundingBox'}, @(area, BB) area / bounding_box2area(BB), 'Interval'),
     ...
    'Smoothing',     create_parameter_structure(inputs.Smoothing, '',
    @imgaussfilt3, 'Function'), ...
    'Median',        create_parameter_structure(inputs.Median, '',
    @medfilt2, 'Function'), ...
    'NumElements',   create_parameter_structure(inputs.NumElements, '',
    @num_regions , 'IntervalFunction'), ...
    'Fill',          create_parameter_structure(inputs.Fill, '', @imclose ,
     'Function'), ...
    'Custom',        inputs.Custom);

%% Defaults
defaults = struct(...
            'Area',             [10, 300], ...
            'Eccentricity',     [0, 0.9], ...
            'AreaPerimeter',    [1, 5], ...
            'Solidity',         [0.6471, 0.9670], ...  % Experimental data
     suggests values around here
            'IgnoreArea',       [40, 300], ...       % Biggest area using
    the gcp-s from mosaikk was a little lss than 250 and smalles was a
    little bigger than 70.
            'Tightness',        [0.4, 0.9], ...
            'Smoothing',        1, ...
            'NumElements',      [0, 10], ...
            'Median',           true, ...
            'Fill',             strel('Disk', 5));

parameters = apply_deafults(parameters, defaults);

% allowed_properties = {'all', 'basic', 'Area', 'Centroid', 'BoundingBox',
     ...
%     'SubarrayIdx', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
    ...
%     'Orientation', 'ConvexHull', 'ConvexImage', 'ConvexArea', 'Image',
    ...
%     'FilledImage', 'FilledArea', 'EulerNumber', 'Extrema', ...
%     'EquivDiameter', 'Solidity', 'Extent', 'PixelIdxList', 'PixelList',
    ...
%     'Perimeter', 'PerimeterOld', 'PixelValues', 'WeightedCentroid', ...
%     'MeanIntensity', 'MinIntensity', 'MaxIntensity', ''};


%% Initialization
necessary_properties = [extract_necessary_properties(parameters), {'Area'
    }];
% This is the properties that will be fetched from regionprops
n_img = numel(images);
right_morphology = false(n_img, 1);

%% Remove morthological incorrect images
if is_images(images)
```

```
    for i = 1:n_img
        img = images{i};

        fields = fieldnames(parameters);
        if in_use(parameters.Smoothing)
            fun = parameters.Smoothing.Function;
            img = fun(img, parameters.Smoothing.Values);
        end

        BW = dist_fun(img) <= max_distance;

        if in_use(parameters.Median)
            fun = parameters.Median.Function;
            BW = fun(BW, 'symmetric');
        end
        if in_use(parameters.Fill)
            BW = parameters.Fill.Function(BW, parameters.Fill.Values);
            BW = imfill(BW, 'Holes');
        end
        satisfies_all = true;
        props = regionprops(BW, necessary_properties);
        for j = 1:numel(fields)
            property = parameters.(fields{j});
            if in_use(property) && usable(property)
                satisfies_all = satisfies_all && apply_constraint(BW,
    property, props);
            end
        end
        right_morphology(i) = satisfies_all;
    end
else  % Binary image
    fields = fieldnames(parameters);
    BW = images; clear images;
     if in_use(parameters.Median)
        fun = parameters.Median.Function;
        BW = fun(BW, 'symmetric');
    end
    if in_use(parameters.Fill)
        BW = parameters.Fill.Function(BW, parameters.Fill.Values);
        BW = imfill(BW, 'Holes');
    end
    props = regionprops(BW, [necessary_properties, 'PixelIdxList']);
    satisfies_all = true(numel(props), 1);
    for j = 1:numel(fields)
        property = parameters.(fields{j});
        if in_use(property) && usable(property)
            [~, s] = apply_constraint(BW, property, props);
            satisfies_all = satisfies_all & s;
        end
    end
    for i = 1:numel(props)
        if ~satisfies_all(i)
            p = props(i);
            idx = p.PixelIdxList;
            BW(idx) = false;
        end
    end
```

```
145     right_morphology = BW;
146 end
147 end
148
149 %%
150 %=========================================================================
151 function res = in_use(prop)
152 res = ~isempty(prop) && prop.Use;
153 end
154
155 %%
156 %=========================================================================
157 function res = usable(prop)
158 res = strcmp(prop.Mode, 'Interval') || strcmp(prop.Mode, 'IntervalFunction
        ');
159 end
160
161 %%
162 %=========================================================================
163 function structure = create_parameter_structure(property_value,
        necessary_properties, fun, mode)
164 structure = struct(...
165     'Use', will_be_used(property_value), ...
166     'Values', property_value, ...
167     'NecessaryProperties', {necessary_properties}, ...
168     'Function', fun, ...
169     'Mode', mode);
170 end
171
172 %%
173 %=========================================================================
174 function structure = apply_deafults( properties, defaults )
175
176 structure = properties;
177 fields = fieldnames(properties);
178 n = numel(fields);
179 for i = 1:n
180     property = fields{i};
181     property_structure = properties.(property);
182     if ~isempty(property_structure) && ...
183         property_structure.Use && ...
184         islogical(property_structure.Values) && ...
185         property_structure.Values
186
187         structure.(property).Values = defaults.(property);
188     end
189 end
190
191 end
192
193 %%
194 %=========================================================================
195 function output_arg = will_be_used( input_arg )
196
197 output_arg = ~islogical(input_arg) || (islogical(input_arg) && input_arg);
198
199 end
```

```
200
201 %%
202 %=========================================================================
203 function [satisfy, satisfies] = apply_constraint(BW, property, stats)
204
205 if strcmp(property.Mode, 'Interval')
206     % NB: Looks only at the largest area
207     n = numel(stats);
208     if n == 1
209         val = evaluate(property, stats);
210         satisfy = is_inside(val, property.Values);
211     elseif n == 0
212         satisfy = false;
213     else
214         vals = zeros(n, 1);
215         satisfies = zeros(n, 1);
216         areas = zeros(n, 1);
217         for i = 1:n
218             vals(i) = evaluate(property, stats(i));
219             satisfies(i) = is_inside(vals(i), property.Values);
220             areas(i) = stats(i).Area;
221         end
222         [~, I] = max(areas);
223         satisfy = satisfies(I(1));
224     end
225 elseif strcmp(property.Mode, 'IntervalFunction')
226     val = property.Function(BW);
227     satisfy = is_inside(val, property.Values);
228 else
229     satisfy = false;
230 end
231
232 end
233
234 %%
235 %=========================================================================
236 function val = evaluate(property, stats)
237     necessary_properties = property.NecessaryProperties;
238     num_props = numel(necessary_properties);
239     field_values = cell(num_props, 1);
240     for i = 1:num_props
241         field_values(i) = {(stats.(necessary_properties{i}))};
242     end
243     val = property.Function(field_values{:});
244 end
245
246 function res = is_inside(val, interval)
247     res = val >= interval(1) && val <= interval(2);
248 end
249
250
251 %%
252 %=========================================================================
253 function res = extract_necessary_properties(parameters)
254
255 res = {};
256 fields = fieldnames(parameters);
```

```
257
258 for i = 1:numel(fields)
259     property = parameters.(fields{i});
260     if ~isempty(property) && property.Use
261         necessary = property.NecessaryProperties;
262         res = [res, necessary];
263     end
264 end
265 res(strcmp('', res)) = [];  % Remove empty strings
266
267 end
268
269 %%
270 %=========================================================================
271 function output_arg = is_image_or_images( input_arg )
272 output_arg = is_binimg(input_arg) || is_image(input_arg) || is_images(
        input_arg);
273 end
```

Script A.56: remove_areas.m: A function that applies a given function to all foreground areas of a given binary image. All areas that gets a value which falls outside a given interval is removed. Unlike Script A.55, this operates on binary images only, and outputs only binary images.

```
1  % This Source Code Form is subject to the terms of the Mozilla Public
2  % License, v. 2.0. If a copy of the MPL was not distributed with this
3  % file, You can obtain one at http://mozilla.org/MPL/2.0/.
4  %            Copyright (c) 2016 Sindre Nistad
5  function BW = remove_areas(binimg, properties, interval, varargin)
6  %% Description
7  % The function removes all areas in the binary image whose areas, or
8  % regions are outside a certain interval relative to a certain property,
9  % or properties. If there is only one property, there is no need to
10 % specify a function, but if there are multiple properties, then a
11 % 'Function' must be specified. The function must take as input a vector
12 % of the same size as there are properties in the cellarray *properties*.
13 % Note, if a single property is given, it can be a normal string.
14 % The flags 'MinBoundary' and 'MaxBoundary' can be set to 'Inclusive' or
15 % 'Exclusive' independent of eachother. The default is 'Inclusive'. This
16 % means that the given *interval* includes the boundaries, and will be
17 % cunted as inside the interval. If 'Exclusive' is chosen for ether end,
18 % the value at the end of the interval will be counted as outside the
19 % allowed interval.
20 %
21 % Leagal calls:
22 % BW = remove_areas(binimg, property, interval)
23 % BW = remove_areas(binimg, property, interval, options )
24 % BW = remove_areas(binimg, properties, interval, 'Function', @(x) ...,
       options )
25 % options are 'MinBoundary', 'MaxBoundary', both of wich have 'Inclusive'
26 % and 'Exclusive' as parameters, while 'Function' takes an arbitrary
27 % function that is subject to two (2) constraints:
28 % 1. The input must be a single vector, whose length is the same as the
29 %    number of properties (can also be one, if there is only one property)
       .
30 % 2. The output of the function must be a scalar.
31
32 %%
```

```
% f : ℝ^n → ℝ


%% Checking input arguments
default = 'Inclusive';

% Defining validation functions

i_p = inputParser;
i_p.FunctionName = 'REMOVE_AREAS';

% Requiered
i_p.addRequired('binimg', @is_binimg);          % Binary image
i_p.addRequired('properties', @is_properties);  % Property / Properties
i_p.addRequired('interval', @is_interval);      % interval to be used

% Optional
i_p.addParameter('MinBoundary', default, @is_boundary);
i_p.addParameter('MaxBoundary', default, @is_boundary);
i_p.addParameter('Function', @(x) x, @is_function);

i_p.parse(binimg, properties, interval, varargin{:});

%% Dealing with the given input data
inputs = i_p.Results;

min_val = interval(1);
max_val = interval(2);

min_boundary = inputs.MinBoundary;
max_boundary = inputs.MaxBoundary;
fun = inputs.Function;

% Is there multiple properties, or a single property?
if ~iscellstr(properties)
    % There is only one property.
    properties = {properties};
end

%% Defining the comperation criteria

outside_interval = make_outside_interval_checker(min_boundary,
    max_boundary);

%% Getting properties
vars = properties;
vars{end + 1} = 'PixelIdxList';

props = regionprops(binimg, vars);

BW = binimg;

for i = 1:numel(props)
    field_vals = zeros(numel(properties), 1);
    element = props(i);

```

```
    for j = 1:numel(properties)
        field_vals(j) = element.(cell2mat(properties(j)));
    end

    val = fun(field_vals);

    if outside_interval(val, min_val, max_val)
        pixels = element.PixelIdxList;
        BW(pixels) = 0;
    end
end

end
```

Script A.57: remove_empty_cells.m: A helper function that removes all empty cells from a cell-array.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function reduced = remove_empty_cells(cellarray)
%% Discription
% REMOVE_EMPTY_CELLS removes all empty cells from a cell array.

%%
reduced = cellarray(~cellfun('isempty',cellarray));

end
```

Script A.58: rmse.m: A utility function that calculates the root mean square error of a given function, usually the transformation of the GCPs in an image by using a ST.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function RMSE = rmse(target, x, fun)

n = size(target, 1);

RMSE = sqrt(1 / n * sum(sum((target - fun(x)).^2)));

end
```

Script A.59: shinji_umeyama.m: An implementation of Umeyama (1991), which uses the sigular value decomposition (SVD) of the covariance matrix.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [R, t, s, RMSE] = shinji_umeyama(A, B, use_horn_scale)
%% Discription
% SHINJI_UMEYAMA implements the algorithm described by Shinji & Umeyama in
% their article "Least-Squares Estimation of Transformation Parameters
% Betweeen Two Point Patterns.
```

```
% It takes as input two n x m matrices of points, and matches the points in
% A to the points in B.
% It then gives the optimal Rotation matrix (R), along with the optimal
% translation vector (t) and the optimal scaling factor (s).
% This method constricts the avilable solution space to rotations,
% translations, and scaling. This method does NOT considere reflection.

%% Error checking
if ~all(size(A) == size(B))
    error('The matrices are of different size. They MUST BOTH be of the size n x m');
end

%% Threshold
% Because floating point numbers are not entielry accurate.

thresh = 1e-5;

%% Decomposition
mu_A = mean(A);
mu_B = mean(B);

n = size(A, 2);

covariance_matrix = 1 / n * bsxfun(@minus, B, mu_B)' * bsxfun(@minus, A, mu_A);

[U, D, V] = svd(covariance_matrix);

n = size(A, 1);
m = size(A, 2);

d = det(U) * det(V);
if sign(d) == 1 && abs(d - 1) < thresh
    S = eye(m);
elseif sign(d) == -1 && abs(d + 1) < thresh
    S = eye(m);
    S(end, end) = -1;
else
    error('There was an error with the decomposition: det(U) * det(V) ~= [-1, 1]');
end

%% Compute statistics
mu_a = mean(A)';
mu_b = mean(B)';

variance_a = 1 / n * sum(sum(bsxfun(@minus,A', mu_a)'.^2));
variance_b = 1 / n * sum(sum(bsxfun(@minus,B', mu_b)'.^2));


%% Compute outputs
R = U * S * V';
s = 1 / variance_a * trace( D * S);
if nargin == 3 && use_horn_scale
    s = sqrt(variance_b/variance_a);
```

```
end
t = mu_b - s * R * mu_a;

RMSE = sqrt(1 / n * sum(sum((B' - bsxfun(@plus, s * R * A', t)).^2)));

end
```

Script A.60: transform_points.m: Implements the function $p' = \boldsymbol{t} + c\boldsymbol{R}p$, where p' is the transformed points, p is the points to be transfored, $\boldsymbol{R}$ is a rotation matrix, $\boldsymbol{t}$ is the translation vector, and c is the scale factor.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function transformed = transform_points(p, R, t, c)

if nargin == 2
    [R, t, c] = extract_parameters_from_similarity_transform(R);
end

if size(R, 1) == 2
    p = p(:,1:2);
end

transformed = bsxfun(@plus, t, c * R * p')';

end
```

Script A.61: verification_algorithm.m: An umbrella function for the different algorithms for finding the absolute orientation parameters.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function [CM, ST, RMSE] = verification_algorithm( CCM, alg, use_horn_scale
    )
%% Discription
% VERIFICATION_ALGORITH implements the verification lagorithm given by Yan
% Li & Ronald Briggs in "Automated Georeferencing Based on Topological
% Point Pattern Matching".
% It takes as input a Collection of Candidate Matchings (the outpu from
% "match_gcps.m", which implements the matching algorithm in the same
% paper.
% The output is the best Candidate Matching along with is Similarity
% Transfom, i.e. the parameters to apply to the image the points was
% extracted from in order to georeference it.
%
% The option *alg* chooses the algorithm to be used when computing the
% optimal absolute orientation. This option is only applicable when
% the 'GetOptimal'-flag is set to true. The valid options are
% 'ShinjiUmeyama', 'Horn', and 'HornHilden'.
% Default is 'Horn'.


i_p = inputParser;
```

```
i_p.FunctionName = 'VERIFICATION_ALGORITHM';

% Requiered
i_p.addRequired('CCM', @is_candidate_point_lists);

% Optional
i_p.addOptional('OrientationAlgorithm', 'Horn',
    @is_valid_orientation_algorithm);

i_p.parse(CCM, alg);

%% Deal with the input
input = i_p.Results;

algorithm = input.OrientationAlgorithm;

%% Initialization

n = size(CCM, 1);
RMSEs = zeros(n, 1);
parameters = cell(n, 1);
%% Compute RMSE

for i = 1:n
    CM = CCM{i};
    m = size(CM, 2) / 2;
    A = CM(:, 1:m);
    B = CM(:, m + 1: 2 * m);
    if strcmpi(algorithm, 'ShinjiUmeyama')
        [R, t, c, RMSE] = shinji_umeyama(A, B, use_horn_scale);
    elseif strcmpi(algorithm, 'HornHilden')
        [R, t, c, RMSE] = horn_hilden(A, B);
    elseif strcmpi(algorithm, 'Horn')
        [R, t, c, RMSE] = horn(A, B);
    else
        error('Invalid choise of algorithm');
    end
    parameters{i} = {real(R), real(t), c};
    RMSEs(i) = real(RMSE);
end

[RMSE, I] = min(RMSEs);
CM = CCM{I, :};
ST = parameters{I, :};

end
```

Script A.62: write_world_file.m: The script writes a world file to a user specified location based on the given ST.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%            Copyright (c) 2016 Sindre Nistad
function write_world_file(path, ST)
%% Discription
% WRITE_WORLD_FILE writes the world file (*.twf, *.jwf, etc.) of the
```

```
% orthophoto. The inputs are the path to where to file is to be written,
% along with its name, the Similarity Transform (ST), e.i. how to get from
    image
% coordinates to world coordinates, and the size of the image, as returned
% from size(orthophoto).

%% Initialization
% Similarity transform parameters, extracted
R = ST{1};
t = ST{2};
s = ST{3};

% Creating a transfomation matrix
n = size(R, 1);
transformation_matrix = eye(4);
transformation_matrix(1:n, 1:n) = R;
transformation_matrix = s * transformation_matrix;
transformation_matrix(1:n, 4) = t;


% Opening the file
fileID = fopen(path, 'w');

% Internal parameter
numer = '%.20f\n';

%% World file contetnt
% Based on the discription from
% http://webhelp.esri.com/arcims/9.3/General/topics/author_world_files.htm
    .

% upper_left_corner = transform_points([1 1 0], R, t, s);
% x_dist = pdist([transform_points([1 2 0], R, t, s); upper_left_corner]);
% y_dist = pdist([transform_points([2 1 0], R, t, s); upper_left_corner]);

A = transformation_matrix(1, 1);
B = transformation_matrix(1, 2);
C = transformation_matrix(2, 4);
D = transformation_matrix(2, 1);
E = transformation_matrix(2, 2);  % Pixels are downward
F = transformation_matrix(1, 4);

%% Write the content to file

fprintf(fileID, numer, abs(A));
fprintf(fileID, numer, D);
fprintf(fileID, numer, B);
fprintf(fileID, numer, -abs(E));
fprintf(fileID, numer, C);
fprintf(fileID, numer, F);

fclose(fileID);
end
```

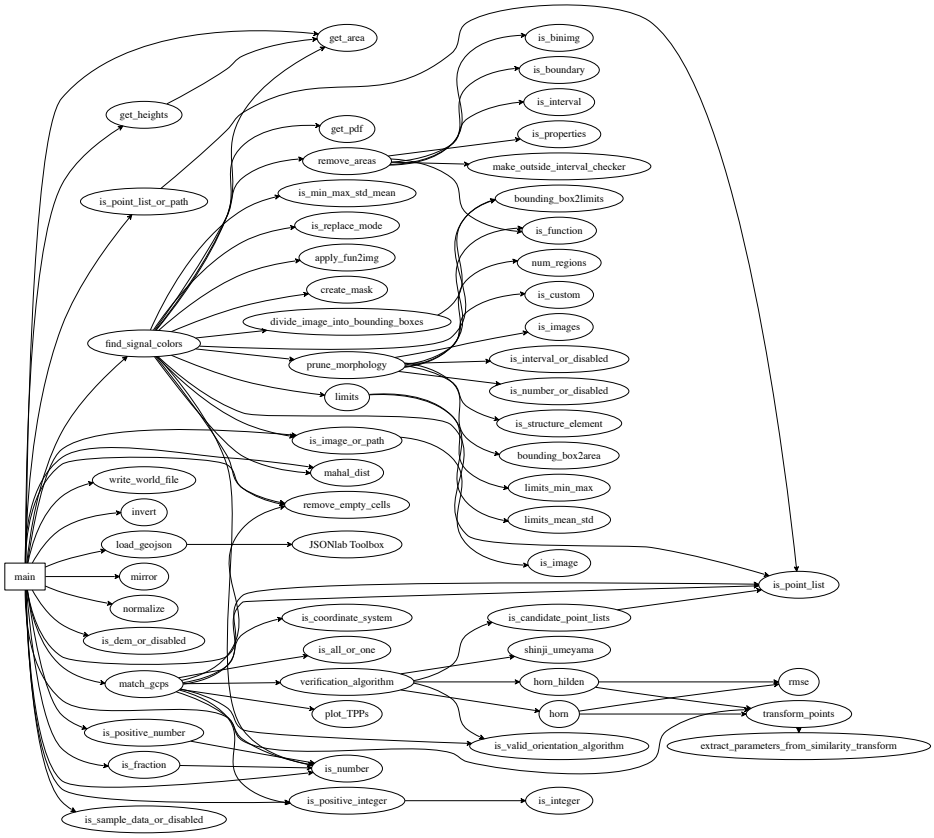


Figure A.1: Shows the dependency graph of "main.m". A directed arrow indicates that the script depends on the script the arrow points at.

A.1.1 Miscellaneous scripts

In this section, the MATLAB scripts that are not used by "main.m" presented. These are included because they where used during the development, and or during testing of the application.

Script A.63: kof2geojson: Converts a set of points in the KOF-format to a set of points in GeoJSON.

```
# -*- coding: utf-8 -*-
"""
This Source Code Form is subject to the terms of the Mozilla Public
License, v. 2.0. If a copy of the MPL was not distributed with this
file, You can obtain one at http://mozilla.org/MPL/2.0/.
            Copyright (c) 2016 Sindre Nistad

This scipt reads a KOF file, and return a geoJSON file with
all the coordinates, and their names of the KOF file.
"""
```

```
__author__ = 'Sindre Nistad'


def kof2geojson(path, crs="EPSG:32632"):
    points = _get_points(path)
    return _geojson_feature_collection(points, crs)


def _get_points(path):
    points = {}
    with open(path, 'r') as f:
        lines = f.readlines()
        for line in lines:
            [num, name, north, east, height] = line.split()
            points[name] = [float(north), float(east), float(height)]
        f.close()
    return points


def _geojson_point(point):
    return "{\"coordinates\": " + str(point) + ", \"type\": \"Point\"}"


def _geojson_feature_collection(points, crs):
    string = "{ \"type\": \"FeatureCollection\", \"features\": ["
    for name in points.keys():
        point = points[name]
        string += _geojson_feature_point(point, name) + ", "
    string += "]"
    if crs != "":
        string += ", \"crs\" : {\"type\": \"name\", \"properties\": {\"
    name\": \"" + crs + "\"}}"
    string += "}"
    return string


def _geojson_feature_point(point, name):
    return "{ \"type\": \"Feature\", \"geometry\": " + _geojson_point(
        point) + ", \"properties\": {\"name\": \"" + name + "\"}}"


def run():
    path = input('Please give the path to the KOF-file: ')
    geojson = kof2geojson(path)
    save_path = input('Please give the path (and name) of where you would
    like to store the result: ')
    f = open(save_path, 'w')
    f.write(geojson)
    f.close()

run()
```

Script A.64: extract_all_gcp: Extracts all the listed GCPs of a specified orthophoto along with a given area around the points. This was used when creating Figure 3.7.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
```

```
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%             Copyright (c) 2016 Sindre Nistad
function extract_all_gcp(img, path, coordinates, area)
if ischar(img)
    img = imread(img);
end


for i = 1:max(size(coordinates))
    name = strcat(path, 'P', num2str(i), '.png');
    extract_area_around_point(img, coordinates(i,:), area, name);
end

end
```

Script A.65: extract_values.m: Extracts all color values from an image that is in a binary mask.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%             Copyright (c) 2016 Sindre Nistad
function RGB_values = extract_values(image, BW)
%% Discription
% Extracts all values that are not 0 when the given image is masked over
% with the mask BW. The result is per band; it gives a matrix of size n x
% m, where m is the number of bands in the image.

%% Consitency check
img_size = size(image);
if any(img_size(1:2) ~= size(BW))
    error('The dimentions of the image and the mask, is inconsistent');
end

%% Initializing
num_bands = img_size(3);
RGB_values = zeros(sum(sum(BW)), num_bands);
idx = BW == 1;

%% Extracting the values
masked_img = mask_image(image, BW);
for i = 1:num_bands
    band = masked_img(:,:, i);
    RGB_values(:, i) = band(idx);
end
```

Script A.66: fuse_gcp.m: A script used to fuse together all the images in a particular folder, and then write the fused image to disk.

```
% This Source Code Form is subject to the terms of the Mozilla Public
% License, v. 2.0. If a copy of the MPL was not distributed with this
% file, You can obtain one at http://mozilla.org/MPL/2.0/.
%             Copyright (c) 2016 Sindre Nistad
img_size = 101;

gcd = zeros(img_size * 4, img_size * 5, 3);
ext = '.png';

```

```
a = ls(strcat(path, '*', ext));
% a = setdiff(strsplit(a, '\n'), '');
tmp = cell(size(a, 1), 1);
for i = 1:size(a, 1)
    tmp{i} = strtrim(a(i, :));
end
a = tmp;
a = sort_nat(a);  % Sorts in a natural order, i.e. file1, file2, file10
location = '';

for i = 1:max(size(a))
    % Using ls, we get a more natural order than dir, but it gives a
    single
    % string wich then must be split. The splitting creates a cell array.
    img_path = strcat(path, cell2mat(a(i)));
    img = im2double(imread(img_path));

    r = ceil(i / 5);
    c = mod(i, 5);
    if c == 0
        c = 5;
    end

    start_row = 1 * r + (img_size - 1) * (r - 1);
    end_row = start_row + (img_size - 1);

    start_col = 1 * c + (img_size - 1) * (c - 1);
    end_col = start_col + (img_size - 1);

    gcd(start_row:end_row, start_col:end_col, :) = img;


    parts = strsplit(img_path, '/');
    name = cell2mat(parts(end));
    location = strcat(location, 'Row ', num2str(r), ', Col ', num2str(c),'
    : ', name, ', ');
end

imwrite(gcd, strcat(path, 'gcp.png'));
```

Appendix B

Data

This appendix list all the data that was used in the thesis. Orthophotos are not included here directly as they are quite large (approximately 4 GB each). Instead, links to where they can be downloaded are provided.

All numbers are in meters, except for indices of images.

B.1 Sample data

The sample data for ground control points that was used by the program in this thesis is available as a .mat file for MATLAB from `https://server.nistad.me/AutoRef/sample-data.mat`, and as a comma separated values (CSV) file from `https://server.nistad.me/AutoRef/sample-data.csv`.

For both of these files, the data is in three columns; red, blue, and green (from left to right), and both sets are normalized with respect to an 8-bit color. That is, an red green blue (RGB) value of 255 is encoded as 1, while an RGB value of 0.

B.2 Ground control points (GCPs) of "Lerkendal"

Table B.1: The measured-in coordinates of the GCPs in the dataset "Lerkendal" in EUREF89-UTM zone 32N (EPSG:32632). All number are in meters.

Name	Northing	Easting	Ellipsoidal height
P1	7032516.8044	570406.8605	80.3184
P2	7032503.4822	570413.3669	80.1282
P3	7032512.5002	570431.9676	80.4122

P4	7032526.0900	570425.3519	80.6571
P5	7032602.5627	570269.4359	79.5778
P6	7032582.0400	570205.1985	83.7770
P7	7032563.9401	570214.1377	83.7857
P8	7032526.8019	570292.1450	75.8363
P9	7032501.2022	570282.1595	75.7164
P10	7032482.3970	570380.5116	78.9091
P11	7032593.3495	570290.8682	79.5930
P12	7032470.1271	570200.9196	72.2874
P13	7032374.9648	570326.0843	78.1933
P14	7032399.5143	570408.4605	81.1564
P15	7032444.1352	570414.4474	81.0613
P16	7032375.5267	570300.6873	77.3698
P17	7032577.5537	570477.3667	87.4470
P18	7032513.9047	570348.3729	75.7412
NEW2	7032518.1030	570306.0950	75.9270
NEW1	7032492.1900	570484.2030	85.8140

B.3 GCPs of “E6”

Table B.2: The measured-in coordinates of the GCPs in the dataset “E6” in EUREF89-UTM zone 32N (EPSG:32632). The heights uses the vertical reference NN2000. The GCPs marked “GUL” All number are in meters. All GCPs marked with an asterisk (*) are visible in the dataset “E6”.

Name	Northing	Easting	Elevation
1	7023477.515	567927.403	150.595
2	7023455.334	568050.074	152.457
3	7023439.758	568138.951	155.682
4	7023499.217	567827.957	149.137
5	7023251.634	567773.382	135.246
6	7023075.583	567621.455	126.821

7	7022893.260	567379.524	120.093
8	7023311.913	567934.724	142.198
9	7023076.764	567791.525	135.378
10	7022863.719	567614.554	120.439
11	7022640.446	567114.808	88.714
12	7023791.602	568012.023	147.629
13	7024111.921	568136.486	154.732
14	7024164.922	568246.363	158.950
15	7023985.359	568165.713	154.643
16	7024359.283	568211.297	162.606
17*	7024963.927	568470.017	162.202
18*	7025385.438	568381.806	159.754
19*	7025890.905	568536.054	157.253
20*	7026037.329	568595.910	158.998
21*	7025971.650	568639.709	151.466
22*	7025166.985	568609.088	153.926
31	7020767.614	564722.506	19.827
32	7019831.582	564520.837	14.634
33	7019967.487	564500.424	12.025
34	7020105.317	564600.959	19.438
35	7020536.896	564688.200	20.074
36	7021522.691	564957.402	21.089
37	7021620.448	564970.565	20.968
41GUL	7022331.719	564898.005	32.813
42GUL	7022371.247	564891.686	29.137
43	7022536.074	565430.822	30.527
44	7022105.009	565667.994	34.494
45	7022183.190	565676.518	32.959
46GUL	7022485.984	566052.290	40.534
48	7022505.642	566441.434	44.684

49	7022566.553	566426.561	44.976

B.4 Sample extracted candidate matching (CM)

Table B.3: The set of CMs that were extracted from "Lerkendal" and used to examine the difference between using (2.31) and (2.38) for the scale factor in the algorithm for absolute orientation proposed by Umeyama (1991).

Northing	Easting	Elevation	Northing	Easting	Elevation	Northing	Easting	Elevation
2393	12879	75.3148117065430	4282	8770	79.8811492919922	12215	9568	79.0053024291992
16276	11578	86.5648040771484	7293	9639	84.4234695434570	18797	8927	82.6201324462891
15965	6863	93.6623840332031	3489	12737	75.0250778198242	8985	13657	81.6264801025391
10142	4255	104.263183593750	15876	2021	96.8410186767578	6149	10757	81.5795745849609
10138	4292	104.271095275879	5645	13951	73.7352905273438	6806	5894	111.660209655762
16268	11531	86.5881042480469	7963	11230	90.3935394287109	8883	13164	82.5342712402344
10908	14483	79.6116027832031	18109	6593	90.9093170166016	12302	12353	83.6191711425781
18609	11588	93.2391128540039	16236	7095	89.7911605834961	4931	10045	72.1892547607422
16272	11479	86.6482620239258	10964	12218	83.1198654174805	5561	12436	75.0390243530273
14008	9759	82.6606140136719	18792	10392	87.4193420410156	18765	10216	93.8134841918945
15956	9217	88.5000839233398	7861	11617	78.4655609130859	4407	9876	71.5133361816406
13482	13715	82.5921325683594	18769	13109	86.7559890747070	5735	5369	112.980842590332
13668	12319	82.8011550903320	7965	11267	94.5992050170898	18725	8206	87.1003799438477
12607	11122	80.9617614746094	5610	5086	106.274963378906	8695	18022	69.0536651611328
16283	11682	86.4423217773438	6128	9488	75.5858078002930	3396	8424	69.6841354370117
16282	11577	86.5587539672852	14417	2551	95.7217025756836	5752	5408	113.090118408203
3933	6031	89.6924972534180	2820	10353	76.7904586791992	13181	3902	102.745635986328
10115	14465	78.7550277709961	14467	2353	95.9849853515625	15198	12956	84.8037109375000
13429	10047	82.3789978027344	8192	4613	116.050262451172	9958	3771	93.1218414306641
18894	9680	92.0177307128906	2901	11433	74.7353744506836	4414	10174	75.2216796875000
17804	12517	90.6737136840820	3300	12678	76.5004577636719	5999	12293	72.4692611694336
5615	11202	72.3039245605469	19351	8685	89.2225494384766	15524	2110	96.4769210815430
15955	2734	98.8096084594727	15989	6972	89.2949829101563	19240	13822	83.5533142089844
14216	10184	82.3425598144531	13969	2150	96.7252502441406	2933	11424	75.3316040039063
2548	9088	70.0598068237305	19162	13442	88.1330413818359	19064	9763	96.3775329589844
11602	10141	78.0420074462891	5089	12492	74.3332519531250	6546	5207	113.764297485352
18061	7187	90.1903915405273	7307	9876	76.9713439941406	17001	10490	87.8917083740234
9845	9739	78.1014709472656	5226	13992	75.8305664062500	7062	10956	74.0719299316406
10278	10007	78.2061462402344	8854	6496	85.3871765136719	4220	4544	91.9881286621094
18760	12128	87.1573333740234	3038	10369	77.2260437011719	2838	10583	69.9070510864258
9534	10539	77.9017105102539	10209	4536	103.556655883789	3994	14119	72.3313293457031
9043	6095	112.749824523926	6065	14240	74.3455505371094	11468	13187	91.7329788208008
6998	11510	73.9162368774414	13929	2190	96.4683685302734	5927	5809	111.346710205078
16230	11435	86.6813583374023	20072	9790	88.5048828125000	16543	8232	88.2557449340820
2124	12073	68.3991622924805	4848	12520	75.1215972900391	4804	10138	73.3374938964844
18912	9671	93.1954345703125	2926	10386	77.3064575195313	3586	8752	72.7491073608398
11666	13249	93.2966232299805	9155	11863	76.1839218139648	18192	10096	89.7292709350586
3300	14071	71.1626892089844	2739	10522	75.0227050781250	2155	10599	67.6751480102539
16323	11619	86.5317535400391	8108	11497	93.2735290527344	4285	4726	93.3028259277344
5239	4884	96.3418502807617	8651	13827	87.2548141479492	9026	3319	91.9000473022461
13617	7887	86.8157958984375	19501	13236	92.7191390991211	13231	3906	102.737976074219
5599	9264	74.6714324951172	13811	2262	93.4580383300781	14642	13267	83.8819885253906
14942	15094	79.5860748291016	7797	11684	77.9755477905273	16943	10511	85.5704498291016
2574	9206	67.8175888061523	20043	9810	88.3042297363281	7728	16294	67.1657257080078
1691	9821	66.6543197631836	12428	12330	83.6968765258789	11761	13115	90.9020004272461
17725	10884	90.0289459228516	11273	13596	82.7702102661133	18363	7178	91.2479095458984

15944	7887	89.9497451782227	10625	13962	76.8690567016602	11281	7789	81.8172531127930
12610	12427	81.6225433349609	11378	13577	83.7250442504883	14529	13336	83.7233047485352
6008	9460	77.7750625610352	15889	1823	97.1462478637695	11988	12457	85.5822372436523
5185	12496	71.6276245117188	12036	9412	85.5272293090820	11717	13006	89.5093078613281
17113	10736	92.8106842041016	5673	14210	73.8860778808594	2299	12682	73.1396026611328
19211	13287	88.1136474609375	12032	13152	93.2754745483398	3468	8793	76.6776428222656
14917	13183	84.3668441772461	6554	3859	94.1248855590820	13904	9229	83.6322402954102
5607	12023	74.1959381103516	16810	9269	86.3732147216797	14981	13127	84.4817657470703
10172	4252	104.240760803223	5581	5042	105.905517578125	15801	2515	96.2978820800781
16869	6205	90.8216857910156	6436	12072	81.3563232421875	7941	11621	78.4559783935547
15851	10818	87.1995773315430	2148	10675	69.1641311645508	8009	6330	106.374130249023
9886	4746	93.1878738403320	14197	3290	93.1219482421875	13099	3896	102.777763366699
14027	2631	92.4377975463867	7613	4995	113.842025756836	18222	9991	89.7305755615234
2976	10464	75.4521636962891	18709	13615	83.9602355957031	8041	6309	106.816543579102
5035	12474	75.3373031616211	2122	12126	68.4417266845703	8591	18583	72.5687866210938
6497	7676	78.0138397216797	11447	13550	82.3288345336914	17755	12693	91.3447418212891
4230	14266	72.0984573364258	2322	9167	68.1504135131836	11120	7733	81.7401809692383
2820	9071	68.0557022094727	2449	10098	67.9443817138672	19523	8960	88.7379760742188
16324	11651	86.4955596923828	16882	6177	91.3350830078125	14655	13368	83.9432601928711
4027	18335	64.2794342041016	19326	8645	89.2252883911133	11431	5169	102.284179687500
18322	11848	92.6246871948242	19347	9810	95.4623336791992	8090	4651	115.435508728027
5171	10846	74.0400619506836	18289	5463	95.5608596801758	16706	10641	88.2966003417969
19598	12247	87.6185531616211	3123	10430	75.6277389526367	18323	10321	87.3389587402344
11768	16606	78.2864837646484	16837	4552	102.615432739258	12370	12351	85.3164138793945
8692	13829	87.1853408813477	2447	8683	77.2466735839844	7442	8504	85.9225769042969
16987	11208	87.8041839599609	15754	10260	88.4813232421875	13962	2190	96.8417510986328
8953	13510	83.2455139160156	11481	13543	83.9155807495117	18355	10392	91.9160003662109
2413	12942	74.9886856079102	12608	12277	86.6295471191406	2424	8920	75.1625442504883
19045	13371	89.1412124633789	9564	10265	78.3708953857422	14474	13372	83.6405639648438
5707	5279	109.991546630859	17013	4105	102.640609741211	2444	12757	76.0265808105469
1891	7089	71.2564315795898	3872	6046	89.0002212524414	10928	3038	91.6831970214844
4157	12443	78.4530334472656	3144	8853	69.8021163940430	3009	10429	76.0832901000977
8903	13594	89.8434753417969	8027	11348	87.5963516235352	4598	11454	75.1763000488281
3142	9174	69.1649017333984	11693	13487	84.3211364746094	4916	8638	77.3755187988281
8755	18584	70.8093414306641	13710	2551	91.9933547973633	3728	6083	87.1908264160156
15014	13589	84.1516876220703	2429	8874	75.6179122924805	3232	14193	70.8361206054688
17047	6307	94.0834121704102	20066	14216	83.3313903808594	3570	8823	71.0979232788086
11065	13669	83.9059066772461	7886	11664	78.3092727661133	2541	9617	74.9198913574219
8710	13873	80.6368103027344	5160	10791	75.5828247070313	15873	1973	96.9096984863281
19201	13412	89.9259643554688	17119	7075	90.2743301391602	7114	5724	112.710449218750
3829	6076	88.0997314453125	3099	14123	70.8812942504883	8942	5769	113.857978820801
17058	4143	102.843299865723	7320	9905	77.3464965820313	7360	9886	92.2328109741211
11855	13459	86.0473937988281	16076	6993	91.2548980712891	15876	1923	96.9973602294922
3384	12719	75.5810165405273	13718	2585	91.9732894897461	8010	11647	78.6041336059570
4340	13974	72.0028610229492	6213	12297	76.5137939453125	6843	6938	90.1780090332031
9299	15521	76.9986572265625	9564	10229	78.4080963134766	3447	12733	75.2401657104492
17114	4174	102.527954101563	12157	12413	86.7376327514648	5612	13965	73.7983093261719
7145	6810	91.9332580566406	2648	14447	69.6886215209961	7487	10230	78.2862014770508
8208	4648	115.648628234863	15491	2114	96.4255828857422	4884	12617	71.1123962402344
4835	12343	79.6679306030273	3884	13923	72.0226135253906	4517	13957	72.5306777954102
19524	14185	82.9745788574219	9143	6257	112.785842895508	16199	8274	91.9279022216797
15764	2006	96.6410369873047	19483	9941	87.4673614501953	8250	3064	91.3441238403320

B.5 Matchings

Table B.4: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'Horn'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, and `'Rematch'`, `true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P5	9129	7371	81.4823837280273	7032602.5627	570269.4359	79.5778
P11	9803	7660	81.6351394653320	7032593.3495	570290.8682	79.593
P6	7129	8011	85.2321929931641	7032582.0400	570205.1985	83.777
P7	7404	8582	85.3965759277344	7032563.9401	570214.1377	83.7857
P8	9845	9737	78.0984497070313	7032526.8019	570292.1450	75.8363
NEW2	10283	10010	78.1958770751953	7032518.1030	570306.0950	75.927
P9	9532	10537	77.8993301391602	7032501.2022	570282.1595	75.7164
P18	11598	10138	78.0417556762695	7032513.9047	570348.3729	75.7412
P12	6997	11508	73.9162368774414	7032470.1271	570200.9196	72.2874
P1	13427	10046	82.3829345703125	7032516.8044	570406.8605	80.3184
P10	12603	11127	80.9545211791992	7032482.3970	570380.5116	78.9091
P4	14009	9756	82.6503906250000	7032526.0900	570425.3519	80.6571
P2	13632	10466	82.1482391357422	7032503.4822	570413.3669	80.1282
P3	14215	10182	82.3428039550781	7032512.5002	570431.9676	80.4122
P17	15636	8148	88.9027709960938	7032577.5537	570477.3667	87.447
P15	13668	12319	82.8011550903320	7032444.1352	570414.4474	81.0613
P16	10114	14464	78.7565765380859	7032375.5267	570300.6873	77.3698
P13	10907	14482	79.6084289550781	7032374.9648	570326.0843	78.1933
NEW1	15849	10817	87.1946792602539	7032492.1900	570484.203	85.814
P14	13481	13714	82.5927429199219	7032399.5143	570408.4605	81.1564

Table B.5: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'HornHilden'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, and `'Rematch'`, `true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P5	9130	7371	81.4818649291992	7032602.5627	570269.4359	79.5778
P11	9803	7660	81.6351394653320	7032593.3495	570290.8682	79.593
P6	7130	8010	85.2291336059570	7032582.04	570205.1985	83.777
P7	7404	8582	85.3965759277344	7032563.9401	570214.1377	83.7857

P8	9845	9737	78.0984497070313	7032526.8019	570292.145	75.8363
NEW2	10283	10010	78.1958770751953	7032518.103	570306.095	75.927
P9	9532	10538	77.8995895385742	7032501.2022	570282.1595	75.7164
P18	11598	10138	78.0417556762695	7032513.9047	570348.3729	75.7412
P12	6997	11508	73.9162368774414	7032470.1271	570200.9196	72.2874
P1	13427	10046	82.3829345703125	7032516.8044	570406.8605	80.3184
P10	12603	11127	80.9545211791992	7032482.397	570380.5116	78.9091
P4	14009	9756	82.650390625	7032526.09	570425.3519	80.6571
P2	13632	10466	82.1482391357422	7032503.4822	570413.3669	80.1282
P3	14215	10183	82.3423461914063	7032512.5002	570431.9676	80.4122
P17	15636	8148	88.9027709960938	7032577.5537	570477.3667	87.447
P15	13668	12319	82.8011550903320	7032444.1352	570414.4474	81.0613
P16	10114	14464	78.7565765380859	7032375.5267	570300.6873	77.3698
P13	10907	14482	79.6084289550781	7032374.9648	570326.0843	78.1933
NEW1	15849	10817	87.1946792602539	7032492.19	570484.203	85.814
P14	13481	13714	82.5927429199219	7032399.5143	570408.4605	81.1564

Table B.6: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'Umeyama'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, `'Rematch'`, `true`, and `'UseHornScale'`, `false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
NEW1	12382	10449	78.9866409301758	7032492.19	570484.203	85.814
P3	11690	10215	78.1325378417969	7032512.5002	570431.9676	80.4122
P4	11573	9985	77.9596786499023	7032526.09	570425.3519	80.6571
P2	11430	10336	78.1380920410156	7032503.4822	570413.3669	80.1282
P10	11008	10571	78.1371612548828	7032482.397	570380.5116	78.9091
P14	11369	11674	79.2441864013672	7032399.5143	570408.4605	81.1564
P18	10557	10182	78.0570907592773	7032513.9047	570348.3729	75.7412
NEW2	9989	10133	78.1413116455078	7032518.103	570306.095	75.927
P8	9844	9973	83.1458282470703	7032526.8019	570292.145	75.8363
P13	10260	12048	77.9012908935547	7032374.9648	570326.0843	78.1933
P9	9679	10339	77.9041824340820	7032501.2022	570282.1595	75.7164
P16	9926	12014	78.4390716552734	7032375.5267	570300.6873	77.3698
P11	9747	9130	78.1569137573242	7032593.3495	570290.8682	79.593
P5	9495	9029	89.6136169433594	7032602.5627	570269.4359	79.5778

P7	8771	9468	89.1512069702148	7032563.9401	570214.1377	83.7857
P12	8616	10795	78.6597671508789	7032470.1271	570200.9196	72.2874
P6	8664	9298	90.2215881347656	7032582.04	570205.1985	83.777

Table B.7: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'Umeyama'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, `'Rematch'`, `true`, and `'UseHornScale'`, `true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P5	9130	7371	81.4818649291992	7032602.5627	570269.4359	79.5778
P11	9803	7660	81.6351394653320	7032593.3495	570290.8682	79.593
P6	7130	8010	85.2291336059570	7032582.04	570205.1985	83.777
P7	7404	8582	85.3965759277344	7032563.9401	570214.1377	83.7857
P8	9845	9737	78.0984497070313	7032526.8019	570292.145	75.8363
NEW2	10283	10010	78.1958770751953	7032518.103	570306.095	75.927
P9	9532	10538	77.8995895385742	7032501.2022	570282.1595	75.7164
P18	11598	10138	78.0417556762695	7032513.9047	570348.3729	75.7412
P12	6997	11508	73.9162368774414	7032470.1271	570200.9196	72.2874
P1	13427	10046	82.3829345703125	7032516.8044	570406.8605	80.3184
P10	12603	11127	80.9545211791992	7032482.397	570380.5116	78.9091
P4	14009	9756	82.650390625	7032526.09	570425.3519	80.6571
P2	13632	10466	82.1482391357422	7032503.4822	570413.3669	80.1282
P3	14215	10183	82.3423461914063	7032512.5002	570431.9676	80.4122
P17	15636	8148	88.9027709960938	7032577.5537	570477.3667	87.447
P15	13668	12319	82.8011550903320	7032444.1352	570414.4474	81.0613
P16	10114	14464	78.7565765380859	7032375.5267	570300.6873	77.3698
P13	10907	14482	79.6084289550781	7032374.9648	570326.0843	78.1933
NEW1	15849	10817	87.1946792602539	7032492.19	570484.203	85.814
P14	13481	13714	82.5927429199219	7032399.5143	570408.4605	81.1564

Table B.8: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'Horn'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, and `'Rematch'`, `false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P4	14008	9759	82.6606140136719	7032526.09	570425.3519	80.6571
P3	14216	10184	82.3425598144531	7032512.5002	570431.9676	80.4122
P1	13429	10047	82.3789978027344	7032516.8044	570406.8605	80.3184
P10	12607	11122	80.9617614746094	7032482.397	570380.5116	78.9091
NEW1	15851	10818	87.1995773315430	7032492.19	570484.203	85.814
P18	11602	10141	78.0420074462891	7032513.9047	570348.3729	75.7412
P15	13668	12319	82.8011550903320	7032444.1352	570414.4474	81.0613
NEW2	10278	10007	78.2061462402344	7032518.103	570306.095	75.927
P14	13482	13715	82.5921325683594	7032399.5143	570408.4605	81.1564
P8	9845	9739	78.1014709472656	7032526.8019	570292.145	75.8363
P9	9534	10539	77.9017105102539	7032501.2022	570282.1595	75.7164
P13	10908	14483	79.6116027832031	7032374.9648	570326.0843	78.1933
P16	10115	14465	78.7550277709961	7032375.5267	570300.6873	77.3698
P12	6998	11510	73.9162368774414	7032470.1271	570200.9196	72.2874

Table B.9: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'HornHilden'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, and `'Rematch'`, `false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P4	14008	9759	82.6606140136719	7032526.09	570425.3519	80.6571
P3	14216	10184	82.3425598144531	7032512.5002	570431.9676	80.4122
P1	13429	10047	82.3789978027344	7032516.8044	570406.8605	80.3184
P10	12607	11122	80.9617614746094	7032482.397	570380.5116	78.9091
NEW1	15851	10818	87.1995773315430	7032492.19	570484.203	85.814
P18	11602	10141	78.0420074462891	7032513.9047	570348.3729	75.7412
P15	13668	12319	82.8011550903320	7032444.1352	570414.4474	81.0613
NEW2	10278	10007	78.2061462402344	7032518.103	570306.095	75.927
P14	13482	13715	82.5921325683594	7032399.5143	570408.4605	81.1564
P8	9845	9739	78.1014709472656	7032526.8019	570292.145	75.8363

P9	9534	10539	77.9017105102539	7032501.2022	570282.1595	75.7164
P13	10908	14483	79.6116027832031	7032374.9648	570326.0843	78.1933
P16	10115	14465	78.7550277709961	7032375.5267	570300.6873	77.3698
P12	6998	11510	73.9162368774414	7032470.1271	570200.9196	72.2874

Table B.10: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm', 'Umeyama', 'RadiusThreshold', 0.05, 'AngleThreshold', 0.05, 'UseProbability', true, 'Rematch', false`, and `'UseHornScale', false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P8	9845	9739	78.10147095	7032526.802	570292.145	75.8363
NEW2	10278	10007	78.20614624	7032518.103	570306.095	75.927
P9	9534	10539	77.90171051	7032501.202	570282.1595	75.7164
P18	11602	10141	78.04200745	7032513.905	570348.3729	75.7412
P7	7442	8504	85.9225769	7032563.94	570214.1377	83.7857
P10	12607	11122	80.96176147	7032482.397	570380.5116	78.9091
P12	6998	11510	73.91623688	7032470.127	570200.9196	72.2874

Table B.11: A table of the candidate matchings (CMs) from calling Script A.1 with the parameters `'OrientationAlgorithm', 'Umeyama', 'RadiusThreshold', 0.05, 'AngleThreshold', 0.05, 'UseProbability', true, 'Rematch', false`, and `'UseHornScale', true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal".

Name	Image row	Image column	Model elevation	Northing	Easing	Ellipsoidal height
P3	14216	10184	82.3425598144531	7032512.5002	570431.9676	80.4122
P4	14008	9759	82.6606140136719	7032526.09	570425.3519	80.6571
P1	13429	10047	82.3789978027344	7032516.8044	570406.8605	80.3184
NEW1	15851	10818	87.1995773315430	7032492.19	570484.203	85.814
P10	12607	11122	80.9617614746094	7032482.397	570380.5116	78.9091
P15	13668	12319	82.8011550903320	7032444.1352	570414.4474	81.0613
P18	11602	10141	78.0420074462891	7032513.9047	570348.3729	75.7412
P14	13482	13715	82.5921325683594	7032399.5143	570408.4605	81.1564
NEW2	10278	10007	78.2061462402344	7032518.103	570306.095	75.927
P8	9845	9739	78.1014709472656	7032526.8019	570292.145	75.8363

P9	9534	10539	77.9017105102539	7032501.2022	570282.1595	75.7164
P13	10908	14483	79.6116027832031	7032374.9648	570326.0843	78.1933
P16	10115	14465	78.7550277709961	7032375.5267	570300.6873	77.3698
P12	6998	11510	73.9162368774414	7032470.1271	570200.9196	72.2874

B.6 Residuals

Table B.12: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm', 'Horn', 'RadiusThreshold', 0.05, 'AngleThreshold', 0.05, 'UseProbability', true`, and `'Rematch', true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.9a, 4.9b, and B.2. The root mean square error (RMSE) of this table is given in Table B.13.

Name	Northing	Easing	Ellipsoidal height
P5	0.0214254427701235	-0.196248529246077	0.582648358354632
P11	-0.0302738044410944	-0.0565632960060611	0.854232372430857
P6	0.0980576416477561	0.0181017303839326	-5.37759045585759
P7	-0.0850275037810206	-0.122028775978833	-5.58203892606279
P8	0.0364821236580610	0.00214479281567037	3.07917905610474
NEW2	-0.0123542640358210	0.0702689943136647	3.11536531995753
P9	0.0358328493312001	-0.0352342003025115	2.41018784435950
P18	0.0611559208482504	-0.117142360075377	4.15022737520420
P12	0.0866151964291930	0.0608434207970277	3.21605242492076
P1	0.0646319817751646	-0.0639430579030886	1.08809408617043
P10	-0.111164638772607	-0.0948854762827978	1.10661771076551
P4	0.0491308197379112	0.0749581315321848	1.37770673822074
P2	-0.0610577240586281	-0.0107493613613769	1.12488470303381
P3	-0.000982397235929966	0.0508796758949757	1.46339710445081
P17	0.0169860003516078	0.142192959552631	-2.92345764038875
P15	-0.0277092205360532	0.0504293909762055	-1.05396206790390
P16	-0.000384650193154812	0.0437309731496498	-1.53737546343288
P13	-0.0319798085838556	0.0287024816498160	-1.77733687057577
NEW1	-0.0542941614985466	0.110284932423383	-3.05395174527419
P14	-0.0550897903740406	0.0442575748311356	-2.26287992447749

Table B.13: An overview of the RMSEs of Table B.12.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	0.0561114854125187	0.0841350974071215	2.74818778032700
Location	0.101129686101091		
Total RMSE	2.75004787037417		

Table B.14: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'HornHilden'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, and `'Rematch'`, `true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.9c, 4.9d, and B.4. The RMSE of this table is given in Table B.15.

Name	Northing	Easing	Ellipsoidal height
P5	0.021392125	-0.17360657	0.583439343
P11	-0.029444232	-0.065066816	0.854355048
P6	0.126592642	0.040469204	-5.376444755
P7	-0.088424392	-0.130633105	-5.582247778
P8	0.036073914	-0.003164913	3.079070878
NEW2	-0.012267631	0.065650636	3.115268718
P9	0.002437494	-0.039524675	2.409270525
P18	0.06316189	-0.120739725	4.150242733
P12	0.08076667	0.056446082	3.215469488
P1	0.069476134	-0.066532408	1.088298263
P10	-0.108250962	-0.096343778	1.106620692
P4	0.055041724	0.072292029	1.37799951
P2	-0.056165135	-0.012569503	1.125061385
P3	-0.027034465	0.048988937	1.462949502
P17	0.026378098	0.138096152	-2.922824438
P15	-0.023923715	0.051457454	-1.053989104
P16	-0.003347485	0.045798352	-1.53798693
P13	-0.033748235	0.031295245	-1.777873599
NEW1	-0.046251245	0.110391949	-3.053598138
P14	-0.052463189	0.047295453	-2.263081344

Table B.15: An overview of the RMSEs of Table B.14.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	0.0582707643834677	0.0828955461579780	2.74801559911539
Location	0.101326963611188		
Total RMSE	2.74988306779328		

Table B.16: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm', 'ShinjiUmeyama', 'RadiusThreshold', 0.05, 'AngleThreshold', 0.05, 'UseProbability', true, 'Rematch', true,` and `'UseHornScale', false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.9e, 4.9f, and B.6. The RMSE of this taable is given in Table B.17.

Name	Northing	Easing	Ellipsoidal height
NEW1	0.298024974	0.815555897	-3.825745861
P3	-2.108291289	1.17517128	0.76968837
P4	1.634812217	-0.895613756	0.631639583
P2	-2.058016484	0.191907824	0.50610372
P10	1.57164172	1.243837651	0.785949409
P14	1.425235148	-0.078322244	-2.254661854
P18	-0.502424667	-0.322618163	3.768764962
NEW2	-0.753562932	-0.689422867	2.795767124
P8	2.628481224	2.431960387	3.249771481
P13	-1.598759569	-1.170220902	-1.546790031
P9	0.817986789	-0.13101498	2.255647272
P16	0.549382042	-0.847950649	-1.142831471
P11	-0.540288381	-3.196484601	0.058046847
P5	-2.062759472	-0.666211403	0.683775932
P7	3.921826009	0.053686198	-5.216306625
P12	-1.864580263	1.056641346	3.550771919
P6	-1.358707055	1.029098982	-5.069590776

Table B.17: An overview of the RMSEs of Table B.16.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	1.75649408491151	1.23377868222613	2.75405656862744
Location	2.14650443909273		
Total RMSE	3.49174868658166		

Table B.18: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'ShinjiUmeyama'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, `'Rematch'`, `true`, and `'UseHornScale'`, `true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.10c, 4.10d, and B.8. The RMSE of this table is given in Table B.19.

Name	Northing	Easing	Ellipsoidal height
P5	0.021392125	-0.17360657	0.583439343
P11	-0.029444232	-0.065066816	0.854355048
P6	0.126592642	0.040469204	-5.376444755
P7	-0.088424392	-0.130633105	-5.582247778
P8	0.036073914	-0.003164913	3.079070878
NEW2	-0.012267631	0.065650636	3.115268718
P9	0.002437494	-0.039524675	2.409270525
P18	0.06316189	-0.120739725	4.150242733
P12	0.08076667	0.056446082	3.215469488
P1	0.069476134	-0.066532408	1.088298263
P10	-0.108250962	-0.096343778	1.106620692
P4	0.055041724	0.072292029	1.37799951
P2	-0.056165135	-0.012569503	1.125061385
P3	-0.027034465	0.048988937	1.462949502
P17	0.026378098	0.138096152	-2.922824438
P15	-0.023923715	0.051457454	-1.053989104
P16	-0.003347485	0.045798352	-1.53798693
P13	-0.033748235	0.031295245	-1.777873599
NEW1	-0.046251245	0.110391949	-3.053598138
P14	-0.052463189	0.047295453	-2.263081344

Table B.19: An overview of the RMSEs of Table B.18.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	0.0582707643834677	0.0828955461579780	2.74801559911539
Location	0.101326963611188		
Total RMSE	2.74988306779328		

Table B.20: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm', 'Horn', 'RadiusThreshold', 0.05, 'AngleThreshold', 0.05, 'UseProbability', true`, and `'Rematch', false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.8a, 4.8b, and B.1. The RMSE of this table is given in Table B.21.

Name	Northing	Easing	Ellipsoidal height
P4	0.012924368	-0.010407407	0.18675861
P3	-0.006570171	0.018034978	0.843267924
P1	0.083604109	-0.048597412	-0.149163553
P10	0.081140718	-0.020975207	0.500091133
NEW1	-0.010006686	0.066202858	-2.02331403
P18	-0.008975131	-0.005441681	1.907362091
P15	0.008825574	-0.04598224	0.222628556
NEW2	0.096613973	-0.080774212	-0.069207165
P14	-0.063579268	-0.043459098	0.357581435
P8	-0.017427854	0.023771937	-0.646713961
P9	-0.027863315	0.040628701	-0.667669827
P13	-0.077383894	-0.027457092	0.089194672
P16	-0.055130178	0.002119996	-0.168899121
P12	-0.016172243	0.132335881	-0.381916764

Table B.21: An overview of the RMSEs of Table B.12.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	0.0801574235541654	0.0972993741020905	2.67544063741710
Location	0.126064986223378		
Total RMSE	2.67840903991424		

Table B.22: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'HornHilden'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, and `'Rematch'`, `false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.8c, 4.8d, and B.3. The RMSE of this table is given in Table B.23.

Name	Northing	Easing	Ellipsoidal height
P4	0.010277015	-0.008416093	0.186692525
P3	-0.009477677	0.019493797	0.843195284
P1	0.081682321	-0.046967569	-0.149211511
P10	0.08024961	-0.020693462	0.500068856
NEW1	-0.014961432	0.066868708	-2.023437966
P18	-0.008608422	-0.003931431	1.907371391
P15	0.006606658	-0.047199717	0.222572928
NEW2	0.098639092	-0.079097158	-0.069156349
P14	-0.065563822	-0.046426384	0.357531527
P8	-0.014860599	0.0257845	-0.646649549
P9	-0.024905702	0.041638328	-0.667595729
P13	-0.076143474	-0.031389296	0.089225389
P16	-0.052896452	-0.001790375	-0.168843545
P12	-0.010037108	0.132126153	-0.38176325

Table B.23: An overview of the RMSEs of Table B.12.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	0.0515579187392759	0.0528176635450803	0.844255142341283
Location	0.0738100573572944		
Total RMSE	0.847475468634218		

Table B.24: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm', 'Umeyama', 'RadiusThreshold', 0.05, 'AngleThreshold', 0.05, 'UseProbability', true, 'Rematch', false`, and `'UseHornScale', false`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.8e, 4.8f, and B.5. The RMSE of this table is given in Table B.25.

Name	Northing	Easing	Ellipsoidal height
P8	20.1727687	3.283412569	4.15472517
NEW2	9.147260046	21.79401123	2.641725851
P9	-13.86443001	-9.518241636	-1.0953322
P18	4.010592187	78.40811232	2.850472558
P7	73.68178166	-98.7256117	3.100910771
P10	-36.97357112	121.8000502	-5.727913987
P12	-56.17440146	-117.041733	-5.924588162

Table B.25: An overview of the RMSEs of Table B.12.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	39.0064711142310	80.1827863713129	3.98373109899465
Location	89.1671689527762		
Total RMSE	89.2561153788469		

Table B.26: A table of the errors of the CMs from calling Script A.1 with the parameters `'OrientationAlgorithm'`, `'Umeyama'`, `'RadiusThreshold'`, `0.05`, `'AngleThreshold'`, `0.05`, `'UseProbability'`, `true`, `'Rematch'`, `false`, and `'UseHornScale'`, `true`. The sample data is the same as described in Section 4.2.1. The set of GCPs is those given in Table B.1. The image points where extracted from the orthophoto "Lerkendal". These residuals are displayed graphically in Figure 4.10a, 4.10b, and B.7. The RMSE of this table is given in Table B.27.

Name	Northing	Easing	Ellipsoidal height
P3	0.010277015	-0.008416093	0.186692525
P4	-0.009477677	0.019493797	0.843195284
P1	0.081682321	-0.046967569	-0.149211511
NEW1	0.08024961	-0.020693462	0.500068856
P10	-0.014961432	0.066868708	-2.023437966
P15	-0.008608422	-0.003931431	1.907371391
P18	0.006606658	-0.047199717	0.222572928
P14	0.098639092	-0.079097158	-0.069156349
NEW2	-0.065563822	-0.046426384	0.357531527
P8	-0.014860599	0.0257845	-0.646649549
P9	-0.024905702	0.041638328	-0.667595729
P13	-0.076143474	-0.031389296	0.089225389
P16	-0.052896452	-0.001790375	-0.168843545
P12	-0.010037108	0.132126153	-0.38176325

Table B.27: An overview of the RMSEs of Table B.12.

RMSE	Northing	Easing	Ellipsoidal height
Per dimension	0.0515579187392759	0.0528176635450803	0.844255142341283
Location	0.0738100573572944		
Total RMSE	0.847475468634218		

B.7 Similarity transforms

All the equations listed in this section are the absolute orientation parameters obtained from the experiments of this thesis.

B.7.1 Lerkendal

Here the absolute orientation parameters obtained from the dataset "Lerkendal" is given. These were obtained by running Script A.1 with the parameters described in Section 4.4.1.

B.7.1.1 Horn

These are the absolute orientation parameters that were used to compute the residuals in Table B.12.

$$\boldsymbol{R} = \begin{pmatrix} -0.000659401553904870 & -0.999762650880823 & -0.0217762965937669 \\ 0.999748766362255 & -0.000171188703487313 & -0.0224137201636997 \\ 0.0224046724309807 & -0.0217856052974617 & 0.999511589755258 \end{pmatrix} \tag{B.1}$$

$$\boldsymbol{t} = \begin{bmatrix} 7032838.77104514 \\ 569977.133606464 \\ 76.1458026023388 \end{bmatrix} \tag{B.2}$$

$$s = 0.0320164691627896 \tag{B.3}$$

B.7.1.2 Horn-Hilden

These are the absolute orientation parameters that were used to compute the residuals in Table B.14.

$$\boldsymbol{R} = \begin{pmatrix} -0.000611895883722124 & -0.999762590978035 & -0.0217804330607695 \\ 0.999748739150053 & -0.000123557683519682 & -0.0224152470737923 \\ 0.0224072343520523 & -0.0217886762880635 & 0.999511465384118 \end{pmatrix} \tag{B.4}$$

$$\boldsymbol{t} = \begin{bmatrix} 7032838.76178280 \\ 569977.107271249 \\ 76.1457901642686 \end{bmatrix} \tag{B.5}$$

$$s = 0.0320170981741677 \tag{B.6}$$

B.7.1.3 Umeyama

These are the absolute orientation parameters that were used to compute the residuals in Table B.16.

$$\boldsymbol{R} = \begin{pmatrix} -0.00624108258281825 & -0.999835608364521 & -0.0170236639576743 \\ 0.999779029528619 & -0.00589718478167606 & -0.0201770990596946 \\ 0.0200733904213614 & -0.0171458291721382 & 0.999651479036065 \end{pmatrix} \tag{B.7}$$

$$\boldsymbol{t} = \begin{bmatrix} 7033283.33399041 \\ 569559.672962777 \\ 70.8421482896873 \end{bmatrix} \tag{B.8}$$

(B.9)

$$s = 0.0751333059427311 \tag{B.10}$$

B.7.1.4 Umeyama with the scale factor of Horn

These are the absolute orientation parameters that were used to compute the residuals in Table B.18.

$$\boldsymbol{R} = \begin{pmatrix} -0.000611895883722124 & -0.999762590978035 & -0.0217804330607695 \\ 0.999748739150053 & -0.000123557683519682 & -0.0224152470737923 \\ 0.0224072343520523 & -0.0217886762880635 & 0.999511465384118 \end{pmatrix} \tag{B.11}$$

$$\boldsymbol{t} = \begin{bmatrix} 7032838.76178280 \\ 569977.107271249 \\ 76.1457901642686 \end{bmatrix} \tag{B.12}$$

$$s = 0.0320170981741677 \tag{B.13}$$

B.7.1.5 Horn without rematching

These are the absolute orientation parameters that were used to compute the residuals in Table B.20.

$$\boldsymbol{R} = \begin{pmatrix} -0.000992564168106669 & -0.999683843351235 & -0.0251242543944417 \\ 0.999825192564730 & -0.000522987255943797 & -0.0186898581383686 \\ 0.0186708095505903 & -0.0251384133714631 & 0.999509610280908 \end{pmatrix} \tag{B.14}$$

$$\boldsymbol{t} = \begin{bmatrix} 7032839.17912822 \\ 569976.903298537 \\ 78.8635042871018 \end{bmatrix} \tag{B.15}$$

$$s = 0.0320405154962622 \tag{B.16}$$

B.7.1.6 Horn-Hilden without rematching

These are the absolute orientation parameters that were used to compute the residuals in Table B.22.

$$\boldsymbol{R} = \begin{pmatrix} -0.000360186361295802 & -0.999941399656667 & 0.0108197746994134 \\ 0.999147844811569 & -0.000806422198091359 & -0.0412666206977161 \\ 0.0412729266503872 & 0.0107956906173952 & 0.999089610095500 \end{pmatrix} \tag{B.17}$$

$$\boldsymbol{t} = \begin{bmatrix} 7032838.68297593 \\ 569977.572377809 \\ 56.3147215300379 \end{bmatrix} \tag{B.18}$$

$$s = 0.0320184544688913 \tag{B.19}$$

B.7.1.7 Umeyama without rematching

These are the absolute orientation parameters that were used to compute the residuals in Table B.24.

$$\boldsymbol{R} = \begin{pmatrix} 0.00636101140034986 & -0.996256633387222 & -0.0862105444010420 \\ 0.999938589704258 & 0.00711945335835071 & -0.00849295026120099 \\ 0.00907492998461047 & -0.0861512264325802 & 0.996240739896706 \end{pmatrix} \tag{B.20}$$

$$\boldsymbol{t} = \begin{bmatrix} 7033267.02554816 \\ 569555.491552381 \\ 130.141591409152 \end{bmatrix} \tag{B.21}$$

$$s = 0.0746425653342246 \tag{B.22}$$

B.7.1.8 Umeyama with the scale factor of Horn, but without rematching

These are the absolute orientation parameters that were used to compute the residuals in Table B.24.

$$\boldsymbol{R} = \begin{pmatrix} -0.000360186360855989 & -0.999941399656625 & 0.0108197744216394 \\ 0.999147844809866 & -0.000806422198252350 & -0.0412666196221286 \\ 0.0412729266903225 & 0.0107956906211634 & 0.999089584865355 \end{pmatrix} \tag{B.23}$$

$$\boldsymbol{t} = \begin{bmatrix} 7032838.68297593 \\ 569977.572377807 \\ 56.3147215784003 \end{bmatrix} \tag{B.24}$$

$$s = 0.0320184544688913 \tag{B.25}$$

B.8 Referencing errors, and residuals

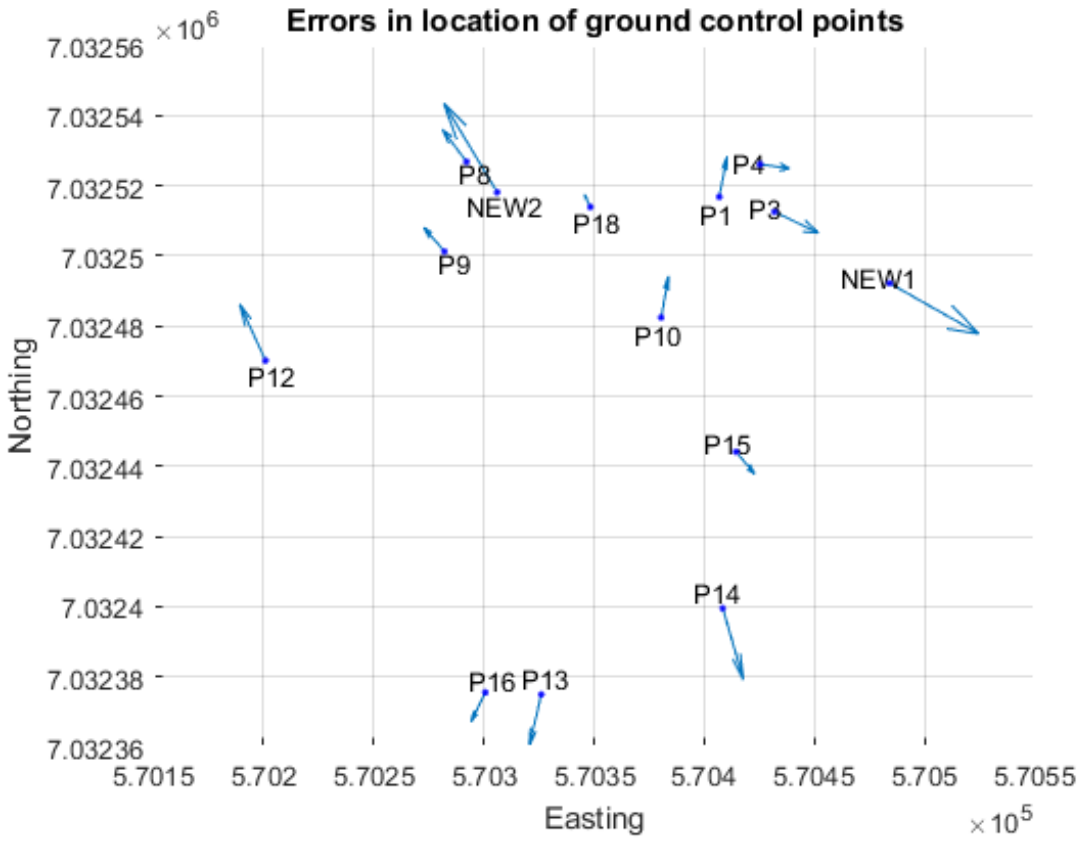
Table B.28: A table showing the greatest, and smallest magnitude of errors for location, elevation, and total. These correspond to Figure 4.8, 4.10a, and 4.10b. In other words, these have *not* been rematched.

Algorithm	Point	Error in	Min/max	Error
Horn	NEW1	Location	max	0.267771
Horn	P18	Location	min	0.028309
Horn	P18	Elevation	max	4.394115
Horn	P18	Elevation	min	1.12447
Horn	P18	Total	max	4.394206
Horn	P1	Total	min	1.126991
Horn-Hilden	P12	Location	max	0.132507
Horn-Hilden	P18	Location	min	0.009464
Horn-Hilden	NEW1	Elevation	max	2.023438
Horn-Hilden	NEW2	Elevation	min	0.069156
Horn-Hilden	NEW1	Total	max	2.024598
Horn-Hilden	P13	Total	min	0.121426
Umeyama	P12	Location	max	129.824187
Umeyama	P9	Location	min	16.817069
Umeyama	P12	Elevation	max	5.92488
Umeyama	P9	Elevation	min	1.095332
Umeyama	P12	Total	max	129.959302
Umeyama	P9	Total	min	16.852702
Umeyama*	P12	Location	max	0.132507
Umeyama*	P18	Location	min	0.009464
Umeyama*	NEW1	Elevation	max	2.023438
Umeyama*	NEW2	Elevation	min	0.69156
Umeyama*	NEW1	Total	max	2.024598
Umeyama*	P13	Total	min	0.121426

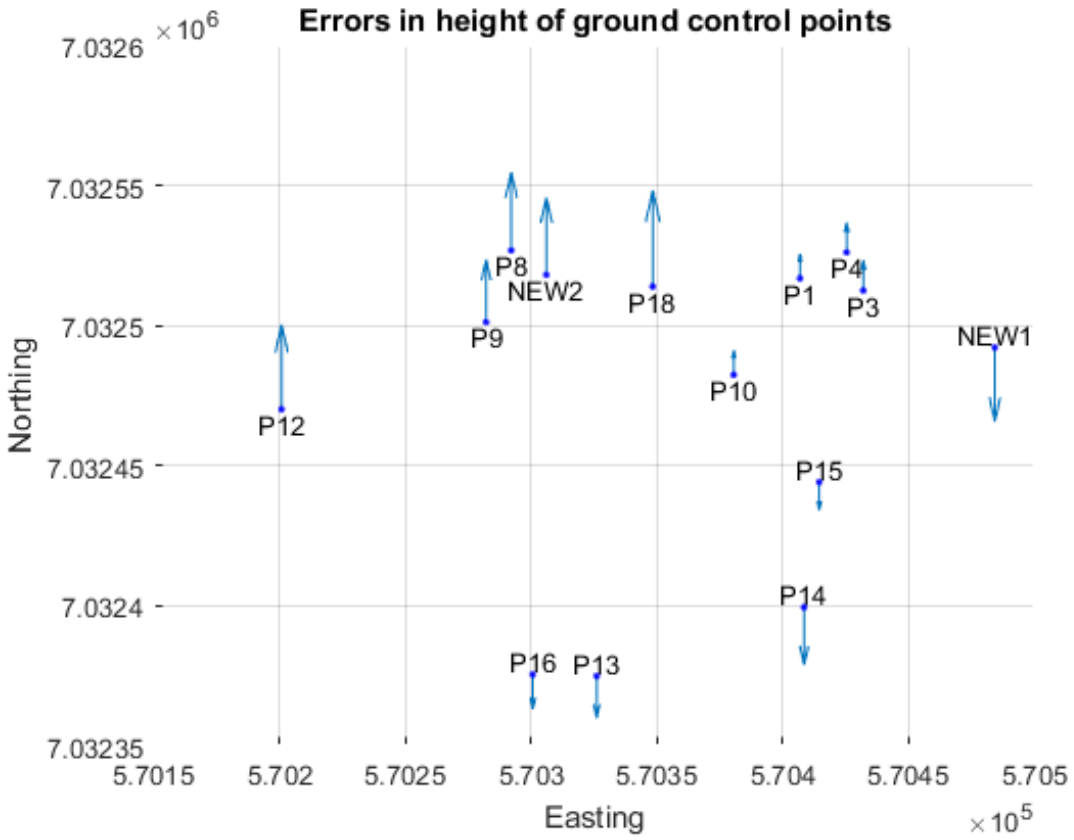
Table B.29: A table showing the greatest, and smallest magnitude of errors for location, elevation, and total. These correspond to Figure 4.9, 4.9e, and 4.9f. In other words, these have been rematched.

Algorithm	Point	Error in	Min/max	Error
Horn	P5	Location	max	0.197415
Horn	P8	Location	min	0.036545

Horn	P7	Elevation	max	5.582039
Horn	P5	Elevation	min	0.582648
Horn	P7	Total	max	5.58402
Horn	P5	Total	min	0.615184
Horn-Hilden	P5	Location	max	0.17492
Horn-Hilden	P8	Location	min	0.036212
Horn-Hilden	P7	Elevation	max	5.582248
Horn-Hilden	P5	Elevation	min	0.583439
Horn-Hilden	P7	Total	max	5.584476
Horn-Hilden	P5	Total	min	0.609096
Umeyama	P7	Location	max	3.922193
Umeyama	P18	Location	min	0.597087
Umeyama	P7	Elevation	max	5.21637
Umeyama	P11	Elevation	min	0.058047
Umeyama	P7	Total	max	6.526366
Umeyama	P16	Total	min	1.52542
Umeyama*	P5	Location	max	0.17492
Umeyama*	P8	Location	min	0.036212
Umeyama*	P7	Elevation	max	5.582248
Umeyama*	P5	Elevation	min	0.583439
Umeyama*	P7	Total	max	5.584476
Umeyama*	P5	Total	min	0.609096

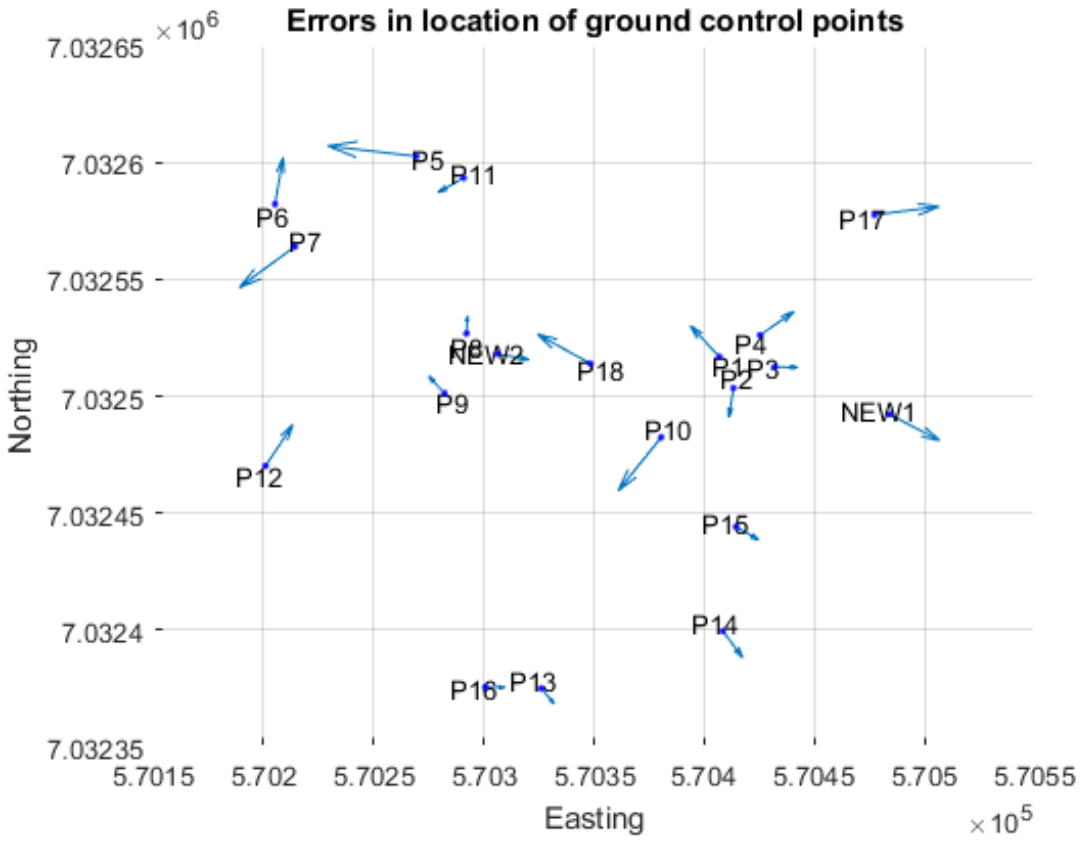


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point NEW1, and has magnitude 0.267771 meters. The smallest error occur at the point P18, with a magnitude of 0.028309 meters.

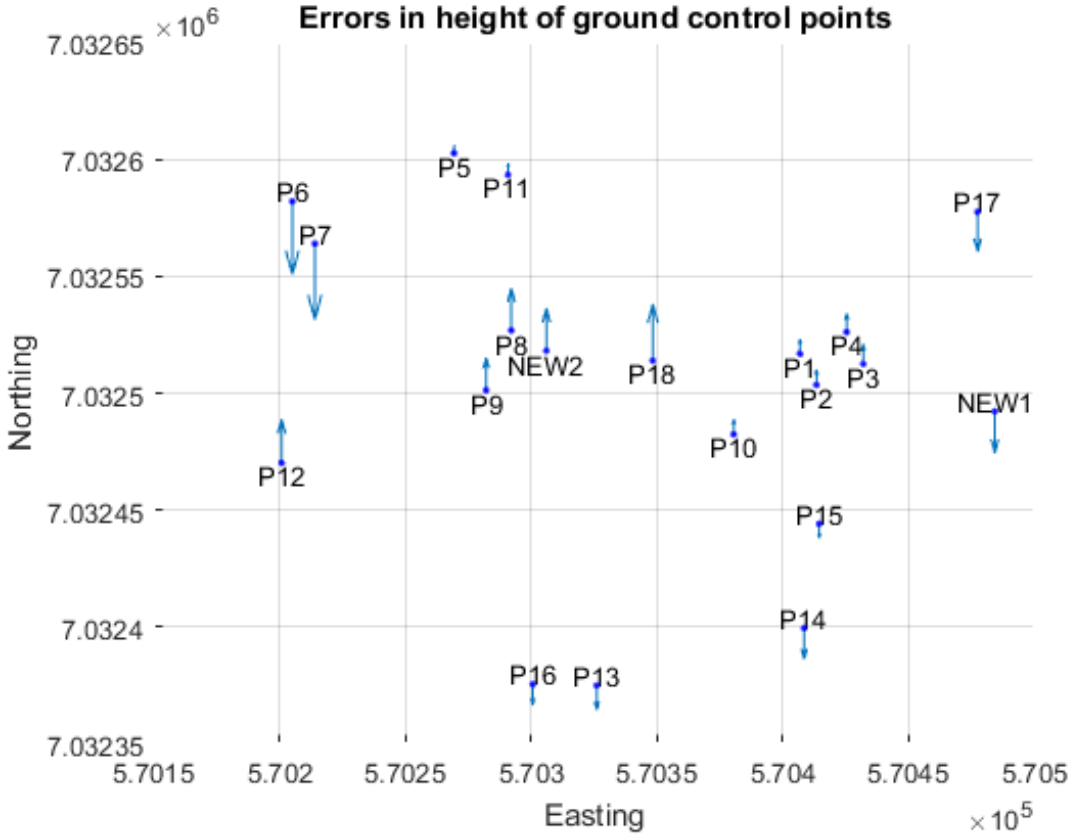


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point P18, and have a magnitude of 4.394115 meters. The smallest has a magnitude of 1.124470 meters at the point P1.

Figure B.1: *Lerkendal - Horn - without rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.14), $\boldsymbol{t}$ by (B.15), and s is given by (B.16). The exact numerical data is given in Table B.20. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 4.394206 meters, and occur at the point P18. The smallest, by contrast, has a magnitude of 1.126991 meters, and occur at the point P1.

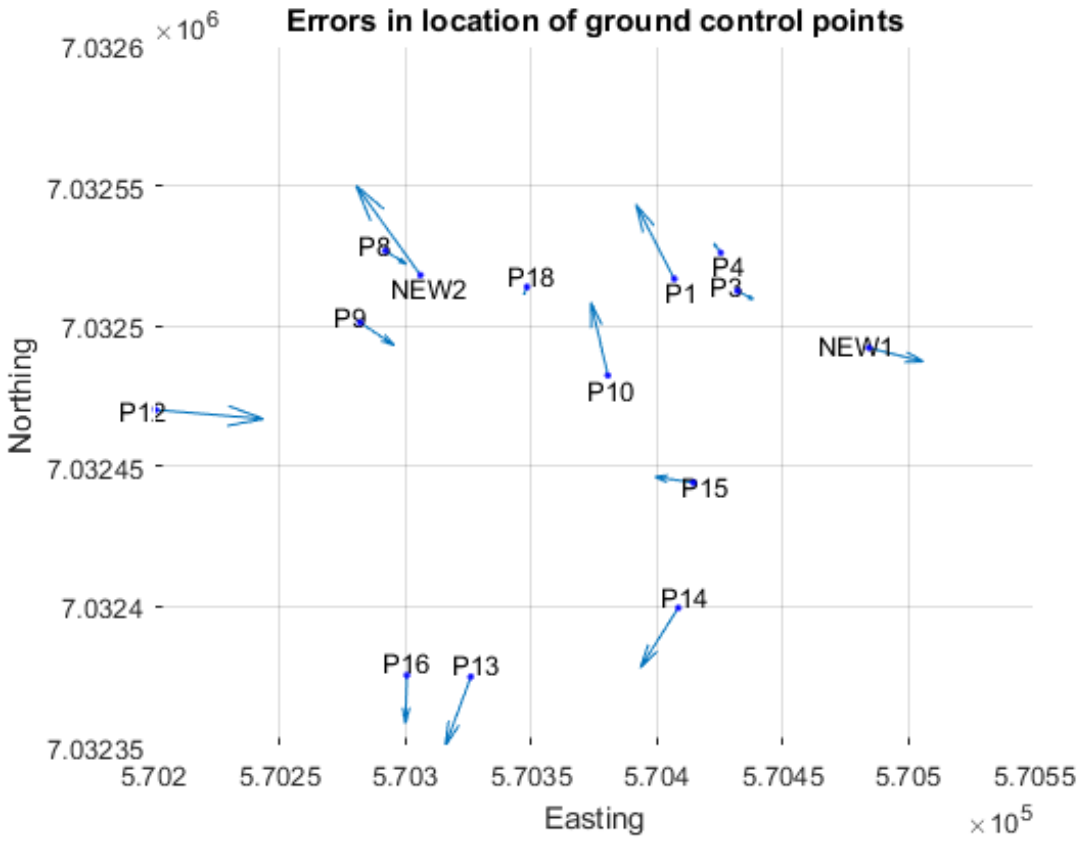


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P5, and has magnitude 0.197415 meters. The smallest error occur at the point P8, with a magnitude of 0.036545 meters.

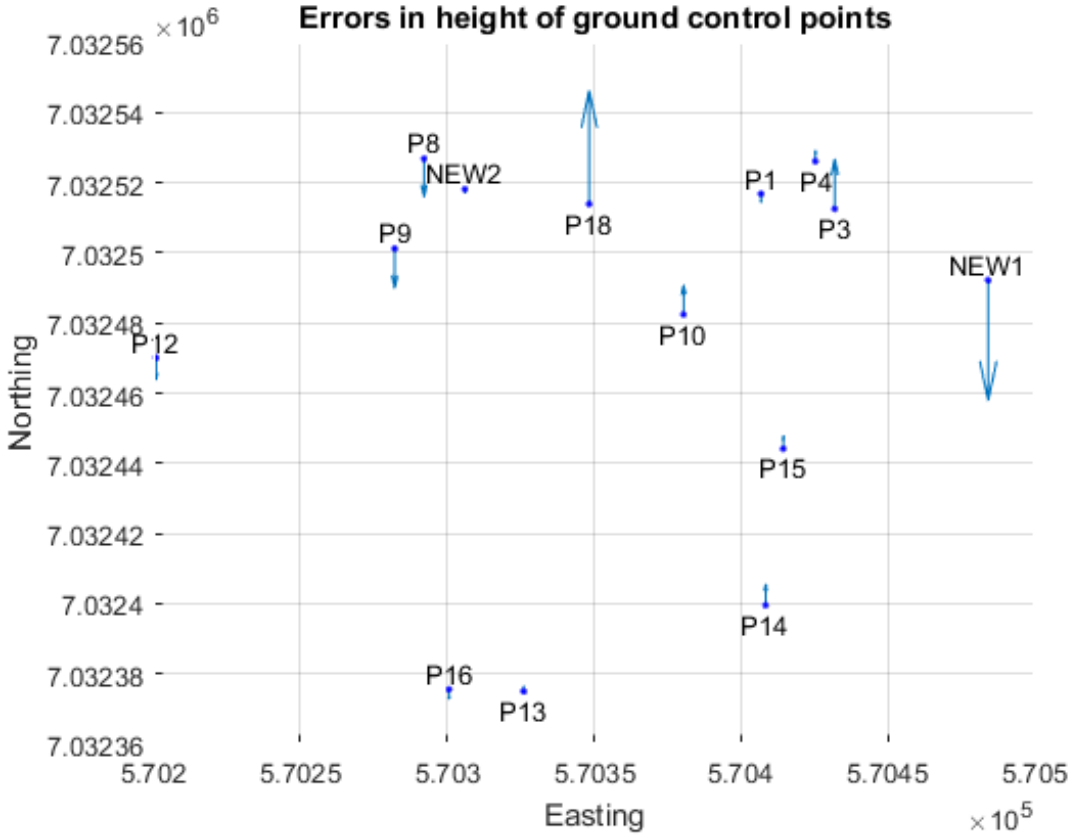


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point P7, and have a magnitude of 5.582039 meters. The smallest has a magnitude of 0.582648 meters at the point P5.

Figure B.2: *Lerkendal - Horn - with rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.1), $\boldsymbol{t}$ by (B.2), and s is given by (B.3). The exact numerical data is given in Table B.12. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 5.584020 meters, and occur at the point P7. The smallest, by contrast, has a magnitude of 0.615184 meters, and occur at the point P5.

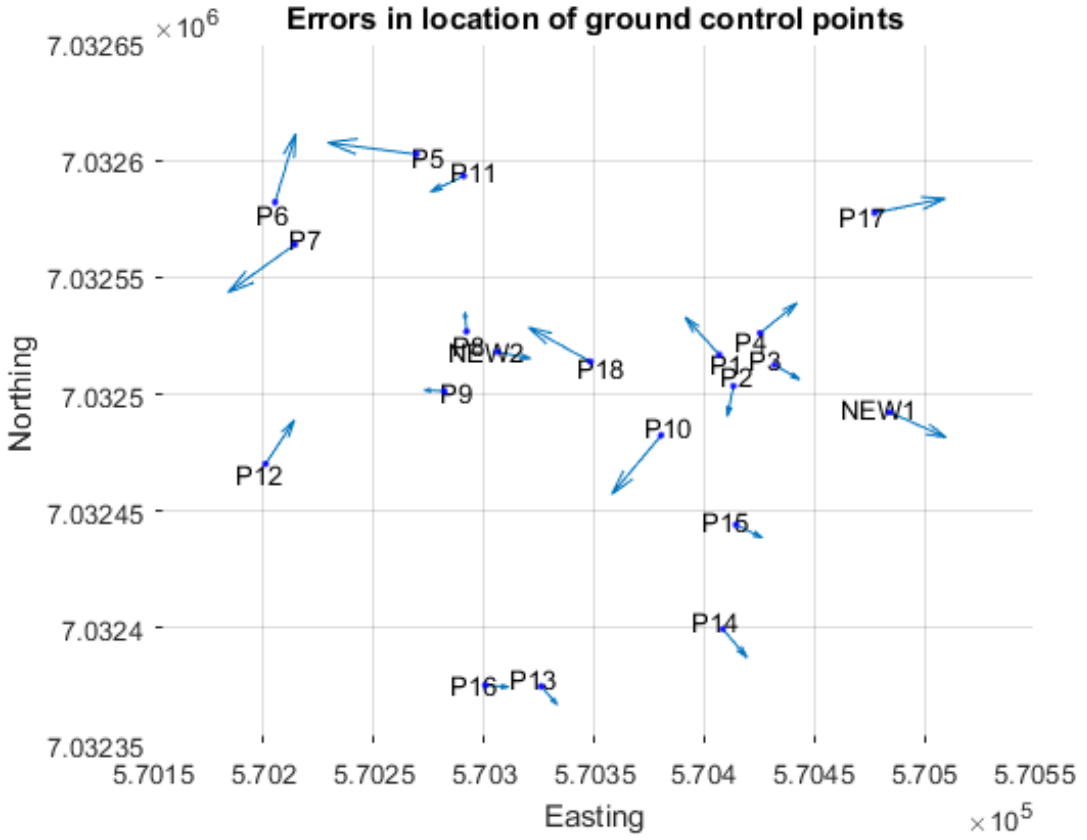


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P12, and has magnitude 0.132507 meters. The smallest error occur at the point P18, with a magnitude of 0.009464 meters.

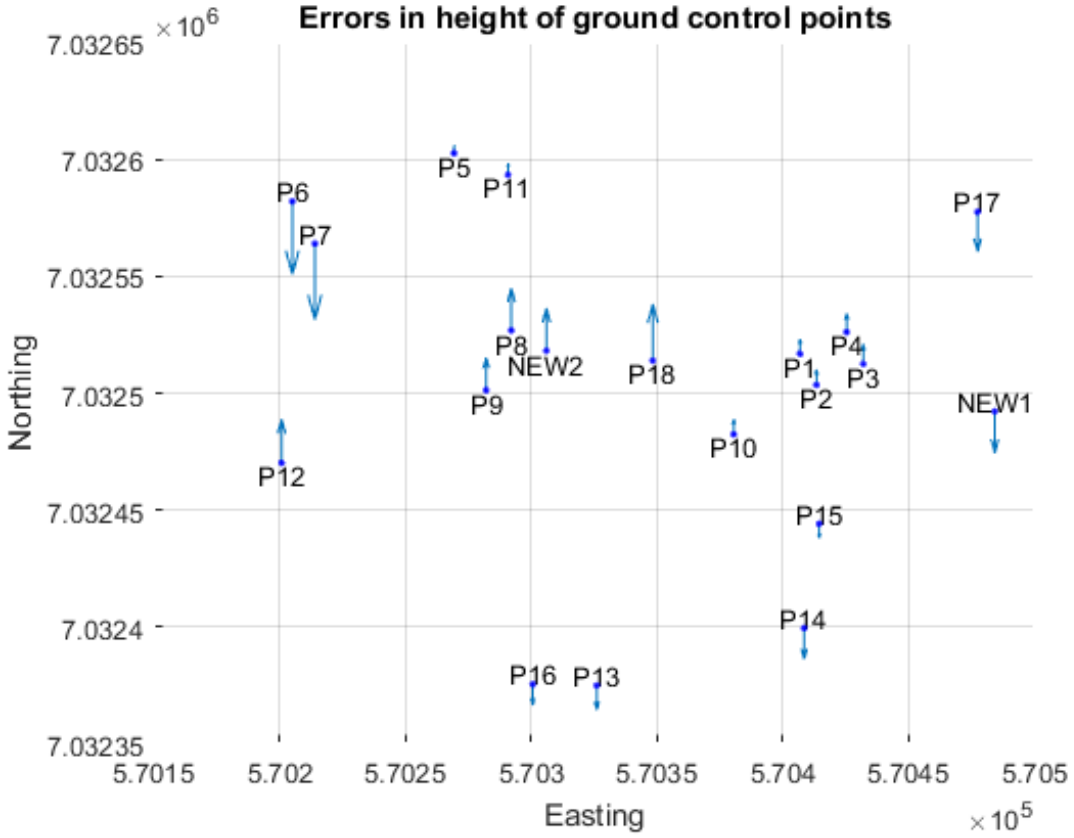


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point NEW1, and have a magnitude of 2.023438 meters. The smallest has a magnitude of 0.069156 meters at the point NEW2.

Figure B.3: *Lerkendal - Horn-Hilden - without rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.17), $\boldsymbol{t}$ by (B.18), and s is given by (B.19). The exact numerical data is given in Table B.22. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 2.024598 meters, and occur at the point NEW1. The smallest, by contrast, has a magnitude of 0.121426 meters, and occur at the point P13.

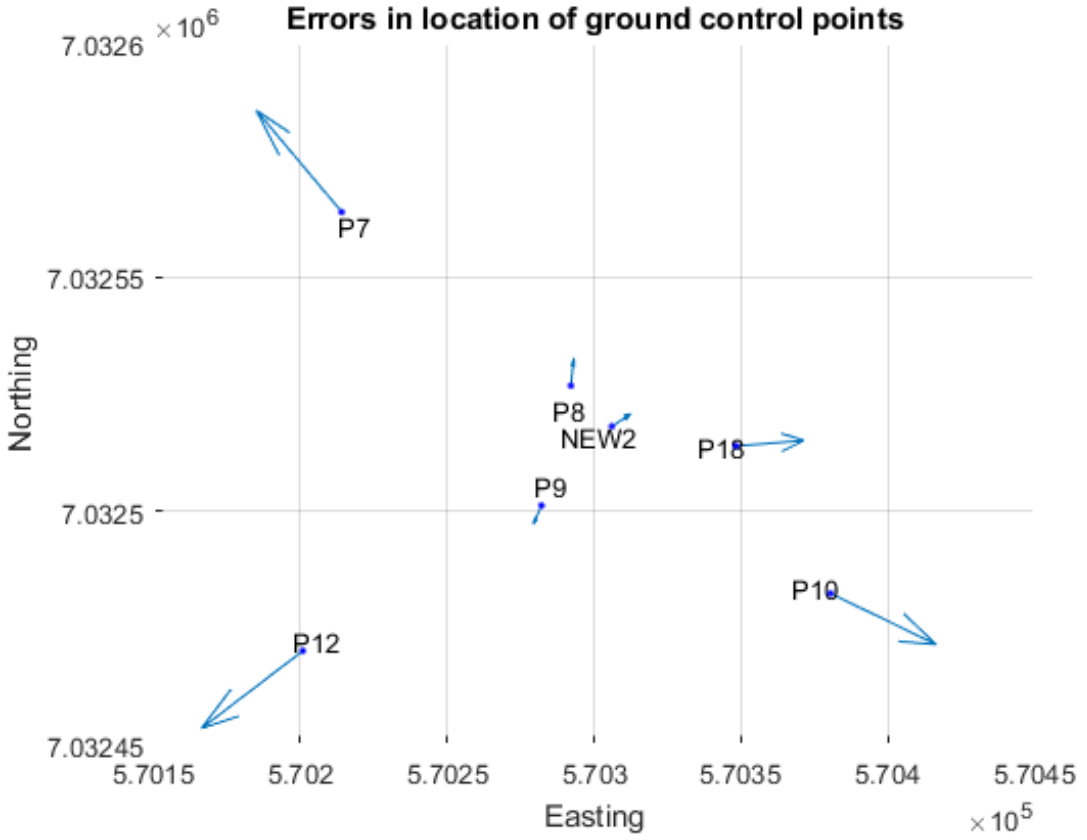


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P5, and has magnitude 0.174920 meters. The smallest error occur at the point P8, with a magnitude of 0.036212 meters.

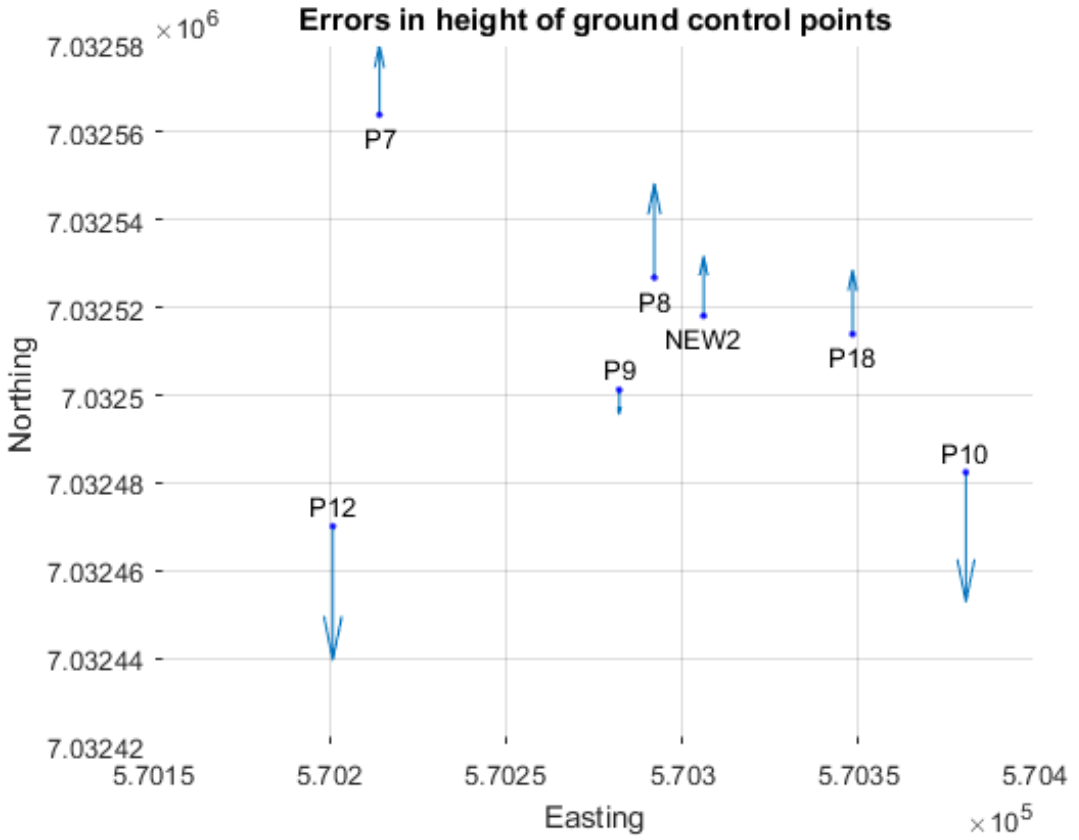


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point P7, and have a magnitude of 5.582248 meters. The smallest has a magnitude of 0.583439 meters at the point P5.

Figure B.4: *Lerkendal - Horn-Hilden - with rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.4), $\boldsymbol{t}$ by (B.5), and s is given by (B.6). The exact numerical data is given in Table B.14. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 5.584476 meters, and occur at the point P7. The smallest, by contrast, has a magnitude of 0.609096 meters, and occur at the point P5.

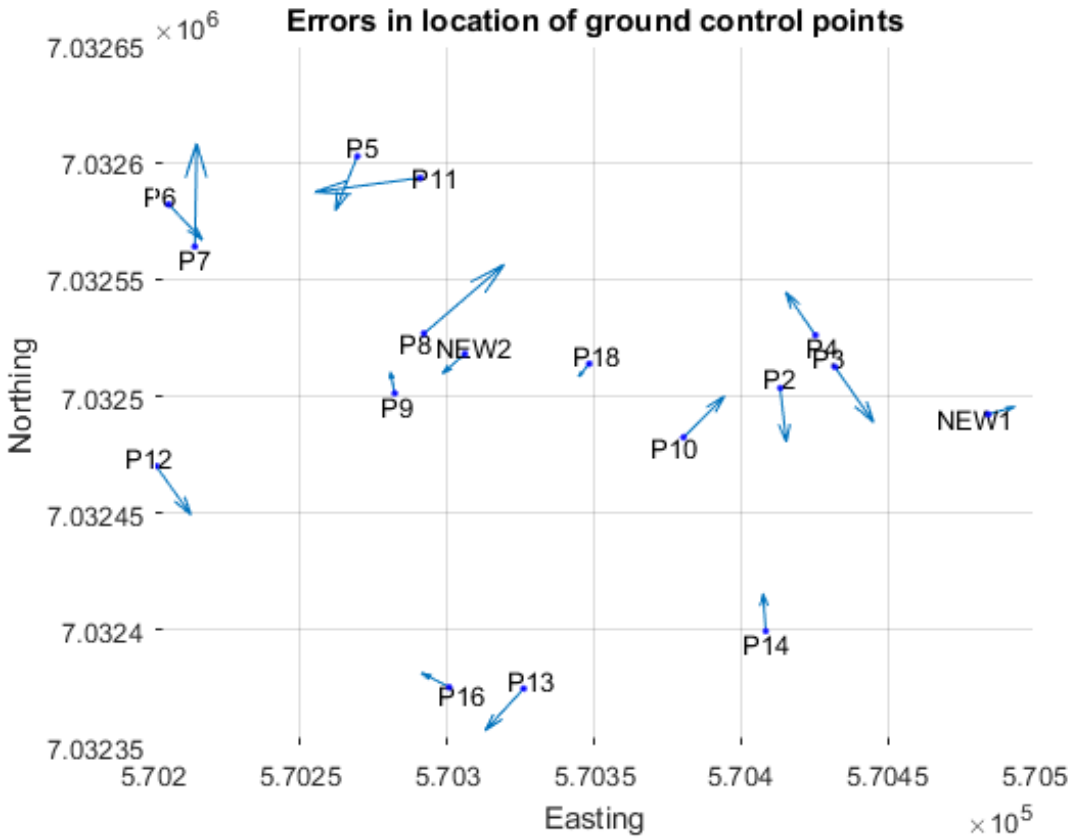


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P12, and has magnitude 129.824187 meters. The smallest error occur at the point P9, with a magnitude of 16.817069 meters.

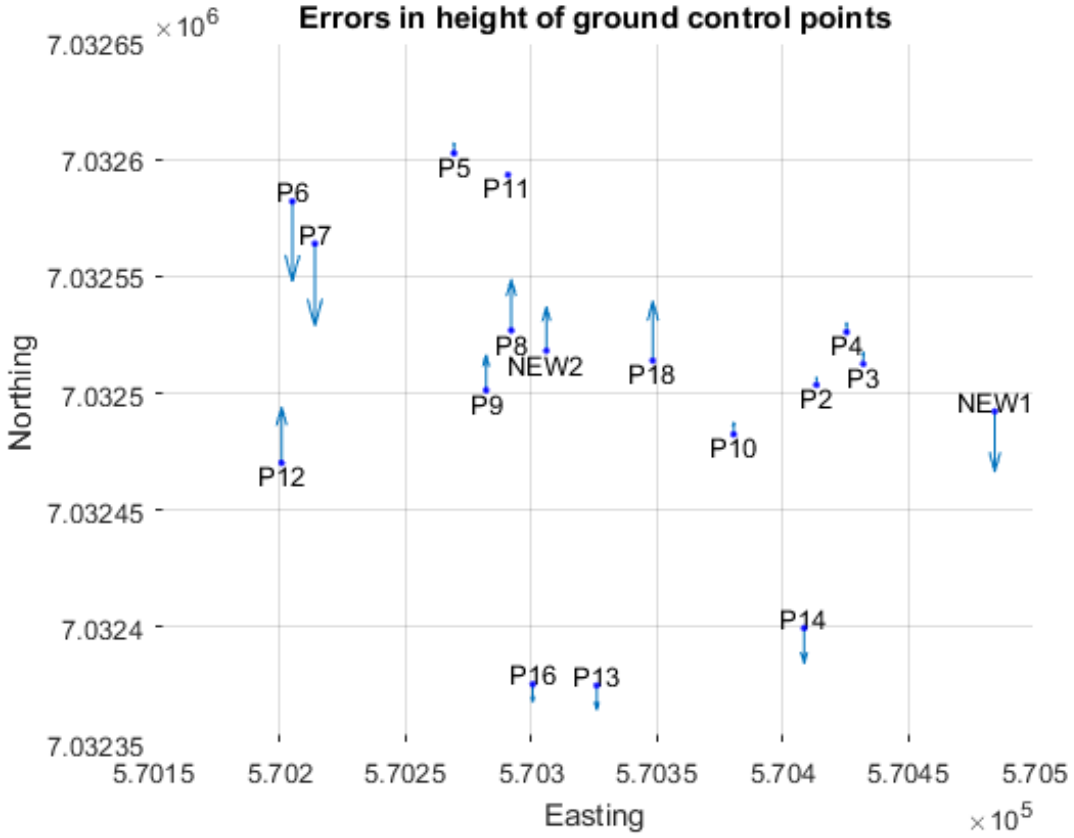


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point P12, and have a magnitude of 5.924588 meters. The smallest has a magnitude of 1.095332 meters at the point P9.

Figure B.5: *Lerkendal - Umeyama - without rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.20), $\boldsymbol{t}$ by (B.21), and s is given by (B.22). The exact numerical data is given in Table B.24. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 129.959302 meters, and occur at the point P12. The smallest, by contrast, has a magnitude of 16.852702 meters, and occur at the point P9.

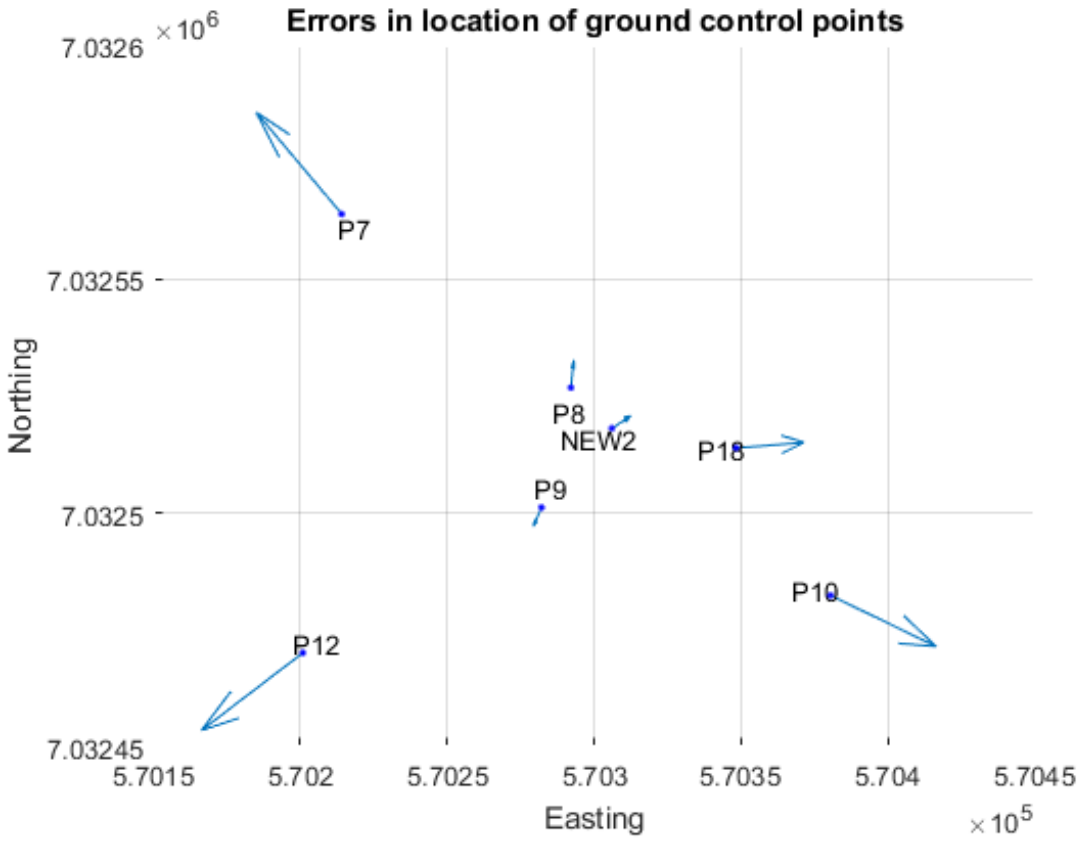


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P7, and has magnitude 3.922193 meters. The smallest error occur at the point P18, with a magnitude of 0.597087 meters.

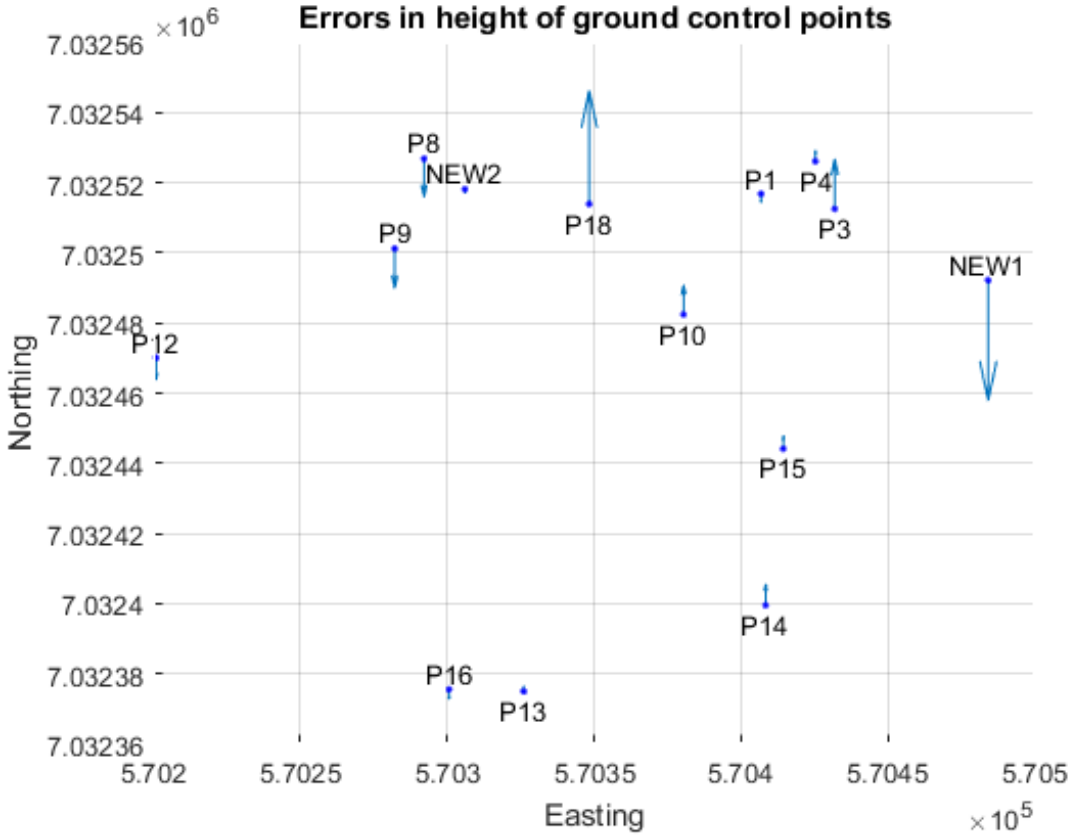


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point P7, and have a magnitude of 5.216307 meters. The smallest has a magnitude of 0.058047 meters at the point P11.

Figure B.6: *Lerkendal - Umeyama - with rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.20), $\boldsymbol{t}$ by (B.21), and s is given by (B.22). The exact numerical data is given in Table B.24. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 6.526366 meters, and occur at the point P7. The smallest, by contrast, has a magnitude of 1.525420 meters, and occur at the point P16.

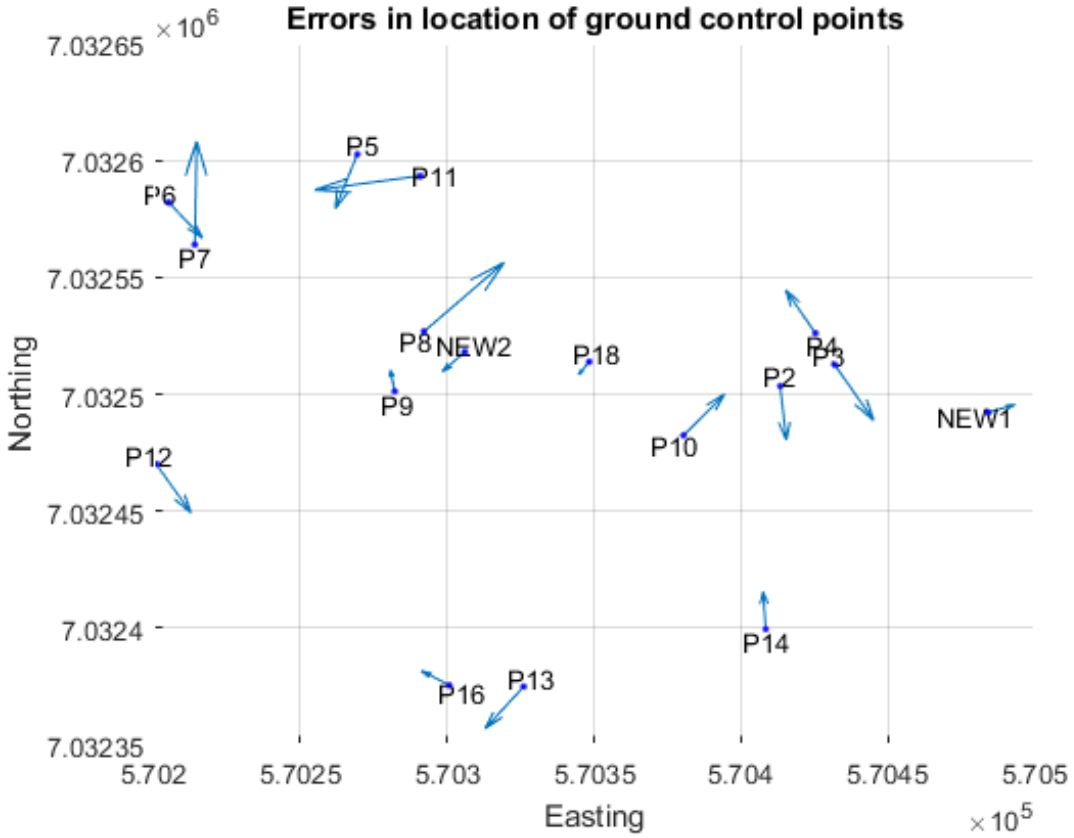


(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P12, and has magnitude 0.132507 meters. The smallest error occur at the point P18, with a magnitude of 0.009464 meters.

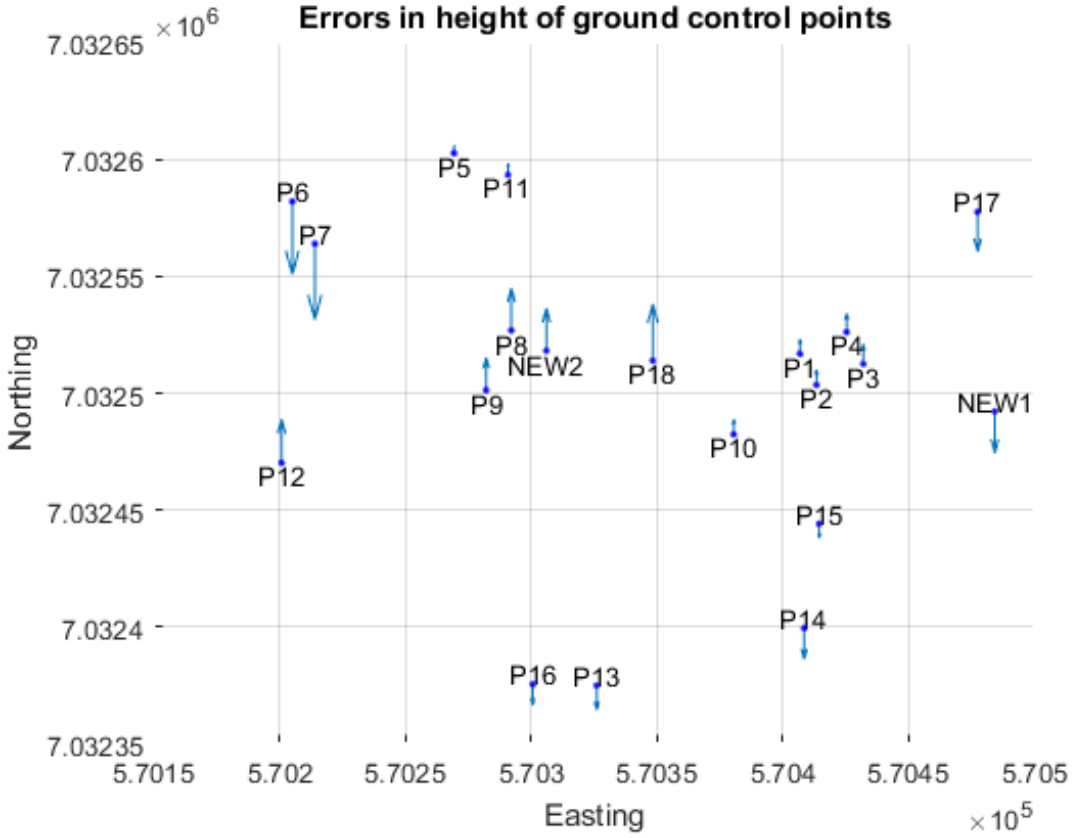


(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point NEW1, and have a magnitude of 2.023438 meters. The smallest has a magnitude of 0.069156 meters at the point NEW2.

Figure B.7: *Lerkendal - Umeyama* - without rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.23), $\boldsymbol{t}$ by (B.24), and s is given by (B.25). The exact numerical data is given in Table B.26. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 2.024598 meters, and occur at the point NEW1. The smallest, by contrast, has a magnitude of 0.121426 meters, and occur at the point P13.



(a) An illustration of the residuals of the location of the GCPs. The largest error in the location is at the point P5, and has magnitude 0.174920 meters. The smallest error occur at the point P8, with a magnitude of 0.036212 meters.



(b) An illustration of the residual error of the elevation of the GCPs. The largest error in elevation occur at the point P7, and have a magnitude of 5.582248 meters. The smallest has a magnitude of 0.583439 meters at the point P5.

Figure B.8: *Lerkendal - Umeyama* - with rematching*: An illustration of the residuals for the location, and elevation of the absolute orientation when $\boldsymbol{R}$ is given in (B.11), $\boldsymbol{t}$ by (B.12), and s is given by (B.13). The exact numerical data is given in Table B.18. The arrows point in the direction of the placement of the estimated Real-World coordinate of the GCP relative to the measured-in coordinate. The largest magnitude of the residuals is 5.584476 meters, and occur at the point P7. The smallest, by contrast, has a magnitude of 0.609096 meters, and occur at the point P5.

Figure B.9: *Lerkendal - Horn - with rematching*: Shows the orthophoto “Lerkendal” . The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Horn (1987) was used to find the absolute orientation. The orange pluses were calculated from the inverse absolute orientation parameters obtained from Horn. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.10: *Lerkendal - Horn - without rematching*: Shows the orthophoto “Lerkendal”. The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Horn (1987) was used to find the absolute orientation. The orange pluses were calculated from the inverse absolute orientation parameters obtained from Horn. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.11: *Lerkendal - Horn-Hilden - without rematching*: Shows the orthophoto “Lerkendal” . The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Horn et al. (1988) was used to find the absolute orientation. The orange pluses were calculated from the inverse absolute orientation parameters obtained from Horn-Hilden. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.12: *Lerkendal - Horn-Hilden - with rematching*: Shows the orthophoto “Lerkendal”. The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Horn et al. (1988) was used to find the absolute orientation. The orange pluses were calculated from the inverse absolute orientation parameters obtained from Horn-Hilden. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.13: *Lerkendal - Umeyama - without rematching*: Shows the orthophoto “Lerkendal” . The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Umeyama (1991) was used to find the absolute orientation. The orange pluses were calculated from the inverse absolute orientation parameters obtained from Umeyama. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.14: *Lerkendal - Umeyama - with rematching*: Shows the orthophoto “Lerkendal” . The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Umeyama (1991) was used to find the absolute orientation. The orange pluses were calculated from the inverse absolute orientation parameters obtained from Umeyama. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.15: *Lerkendal - Umeyama* - without rematching*: Shows the orthophoto "Lerkendal" . The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Umeyama (1991) was used to find the absolute orientation. Instead od using the porposed scale factor, however, the scale factor is calculated as proposed in Horn (1987). The orange pluses were calculated from the inverse absolute orientation parameters obtained from Umeyama*. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

Figure B.16: *Lerkendal - Umeyama* - with rematching*: Shows the orthophoto "Lerkendal" . The blue crosses indicate the areas considered to be a GCP by the prototype when the algorithm proposed by Umeyama (1991) was used to find the absolute orientation. Instead od using the porposed scale factor, however, the scale factor is calculated as proposed in Horn (1987). The orange pluses were calculated from the inverse absolute orientation parameters obtained from Umeyama*. The measured-in coordinates of the CM where then inverted to image coordinates and displayed as orange pluses.

B.9 Distance metrics applied to “Lerkendal”

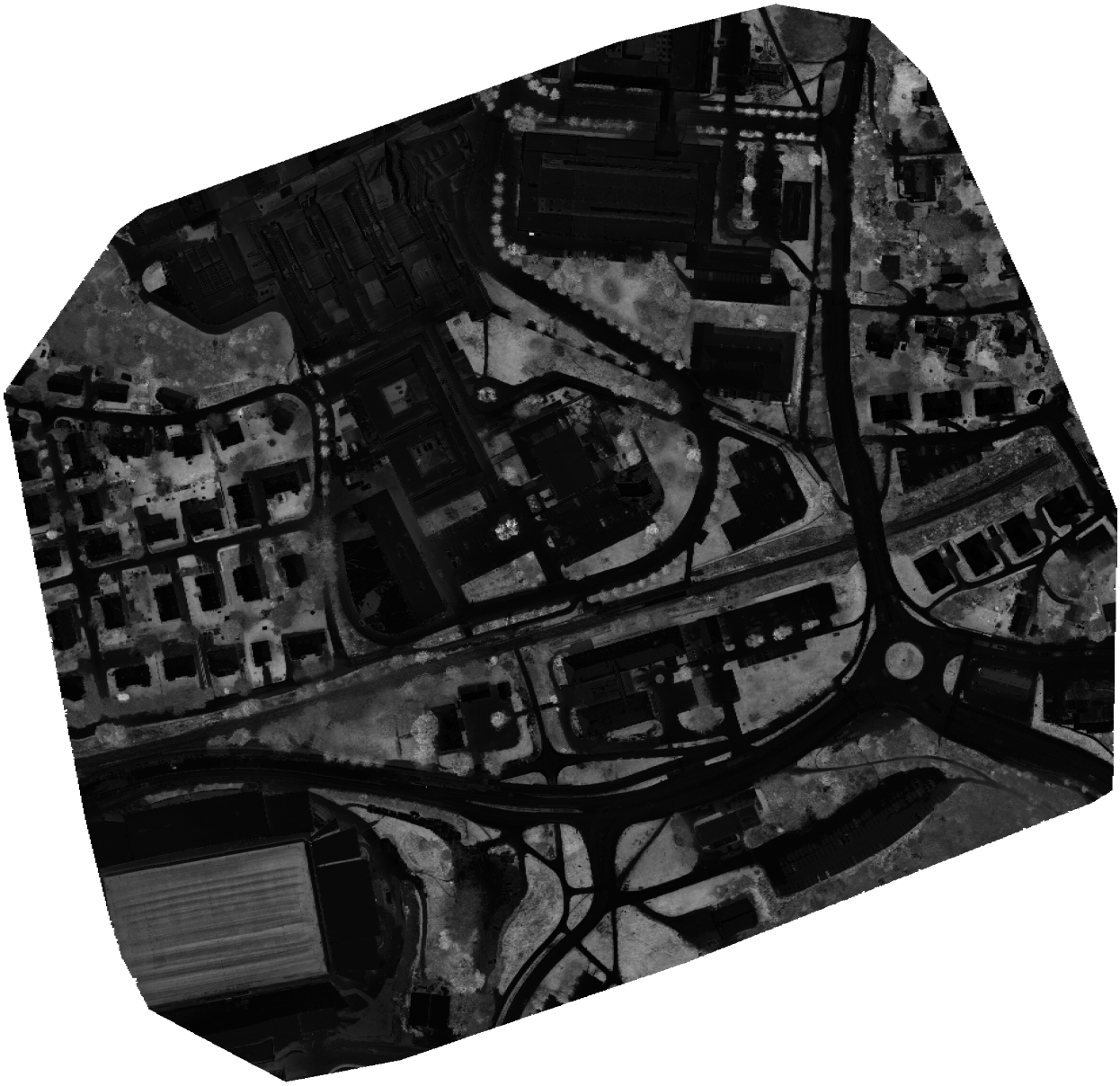


Figure B.17: Using reference data from the orthophoto

Figure B.18: Using the reference data from a marked GCP

Appendix C

Mozilla Public License Version 2.0