TAMPERE UNIVERSITY OF TECHNOLOGY

AKSELI PALÉN

ADVANCED ALGORITHMS FOR MANIPULATING 2D OBJECTS ON TOUCH SCREENS

Master of Science Thesis

Examiner: Adj. Prof. Ossi Nykänen

Examiner and topic approved in the Faculty Council meeting of Faculty of Computing and Electrical Engineering on December 9, 2015

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology

PALÉN, AKSELI: Advanced algorithms for manipulating 2D objects on touch screens
Master of Science Thesis, 59 pages
May 2016
Major: Hypermedia
Examiner: Adj. Prof. Ossi Nykänen

Keywords: multi-touch gestures, affine transformation, similarity, rigid body, direct manipulation, pattern recognition, usability, human-computer interaction

Multi-touch gestures are widely used on touch screens to scale, rotate, or otherwise transform geometric objects such as maps and images. Typical gesture recognition implementations handle one or two touch points successfully but work in various unnatural and error-prone ways with additional fingers or users which are increasingly common due to growing screen sizes. We claim this deficiency to originate from the lack of software developer friendly material for estimating transformations from unlimited and changing number of touch points. We attempt to correct that by deriving 7 reliable and fast algorithms to estimate an optimal translation, rotation, scaling, or one of their combinations from any number of touch points. Mathematically this reduces to solving an optimal nonreflective similarity transformation matrix through the least squares method for each combination. We present the algorithms in implementation-ready source code, show them to be computationally efficient, and implement them in a production-ready software package called Nudged.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma

PALÉN, AKSELI: Edistyneitä algoritmeja kaksiulotteisten kappaleiden muunteluun kosketusnäytöllä
Diplomityö, 59 sivua
Toukokuu 2016
Pääaine: Hypermedia
Tarkastaja: dosentti Ossi Nykänen

Avainsanat: kosketuseleet, affiini muunnos, similaarisuus, jäykkä kappale, suorakäyttöliittymä, hahmontunnistus, käytettävyys

Monikosketusohjatut geometriset muunnokset ovat yleinen tapa skaalata, kiertää ja siirtää kosketusnäytöllä esitettäviä geometrisia kappaleita kuten karttoja ja kuvia. Tyypilliset muunnoseleen tunnistavat toteutukset ovat suunniteltu yhdelle tai kahdelle sormelle ja reagoivat vaihtelevin ja virheherkin tavoin näyttöjen kasvaessa yleistyviin useampiin sormiin tai käyttäjiin. Väitämme tämän ongelman johtuvan julkisen, toteuttajille suunnatun lähdemateriaalin puutteesta koskien muunnosten estimointia rajoittamattomasta ja muuttuvasta määrästä kosketuspisteitä. Pyrimme täyttämään puuttuvan aukon johtamalla luotettavat ja nopeat algoritmit laskemaan optimaalisen skaalauksen, kierron, siirron tai näiden yhdistelmän useasta kosketuspisteestä. Matemaattisesti tämä pelkistyy optimaalisen, ei-peilaavan similaarimuunnoksen ratkaisemiseksi jokaiselle yhdistelmälle pienimmän neliösumman menetelmää käyttäen. Esitämme algoritmit toteutusvalmiina lähdekoodina, osoitamme niiden olevan laskennallisesti tehokkaita ja tarjoamme ne tuotantovalmiissa ohjelmistopaketissa nimeltä Nudged.

PREFACE

This master's thesis is made for the Faculty of Computing and Electrical Engineering at Tampere University of Technology.

I would like to give my deep regards to my supervisor, Adjunct Professor Ossi Nykänen (Tampere University of Technology), and my project manager, Adjunct Professor Jukka Leppänen (Infant Cognition Laboratory at University of Tampere) for all their support. I would also like to thank M.Sc. Olli Suominen and Associate Professor Atanas Gotchev (3D Media Research Group at Tampere University of Technology) for generously allowing me to borrow their group's large touch screen.

In Tampere, Finland, on May 17, 2016

Akseli Palén

CONTENTS

1 Introduction **1**

2 Background **3**
- 2.1 Multi-touch interaction . . . 3
- 2.2 Geometric transformations . . . 4
- 2.3 Multi-touch transforming . . . 5
- 2.4 N-pointer approach . . . 7

3 Method **10**
- 3.1 Problem formulation . . . 10
- 3.2 Transformation groups . . . 12
  - 3.2.1 Basic groups . . . 12
  - 3.2.2 Composite groups . . . 13
  - 3.2.3 Pivoted groups . . . 14
- 3.3 Least squares estimation . . . 15

4 Algorithms and derivation **17**
- 4.1 Translation estimation . . . 17
  - 4.1.1 Special cases . . . 18
  - 4.1.2 Algorithm . . . 18
- 4.2 Scaling estimation . . . 19
  - 4.2.1 Special cases . . . 21
  - 4.2.2 Algorithm . . . 21
- 4.3 Rotation estimation . . . 23
  - 4.3.1 Special cases . . . 25
  - 4.3.2 Algorithm . . . 26
- 4.4 Translation-scaling estimation . . . 27
  - 4.4.1 Special cases . . . 29
  - 4.4.2 Algorithm . . . 30
- 4.5 Translation-rotation estimation . . . 32
  - 4.5.1 Special cases . . . 34
  - 4.5.2 Algorithm . . . 34
- 4.6 Scaling-rotation estimation . . . 35
  - 4.6.1 Special cases . . . 37
  - 4.6.2 Algorithm . . . 37
- 4.7 Translation-scaling-rotation estimation . . . 38
  - 4.7.1 Special cases . . . 41
  - 4.7.2 Algorithm . . . 41

5 Applications **43**
5.1 Multi-touch . . . 43
5.1.1 Concepts . . . 43
5.1.2 Applicable devices and interfaces . . . 44
5.1.3 Challenges . . . 45
5.1.4 Solution . . . 46
5.2 Spatial correction . . . 47
5.3 Geometric layout . . . 48

6 Analysis and discussion **50**
6.1 Computational efficiency . . . 50
6.1.1 Time and space complexity . . . 50
6.1.2 Running time on web browsers . . . 51
6.2 Robustness . . . 53

7 Conclusion **55**

References **56**

LIST OF FIGURES

1.1 A 4-finger composite rotation, scaling, and translation on a real device, made possible by the results of this thesis. . . . 1

2.1 The pinch zoom is often used on touch devices to scale geographic maps. The map shrinks when the index finger and the thumb are moved closer to each other during a pinch-like gesture. . . . 3

2.2 A piece of black cardboard on a slick wooden table. . . . 4

2.3 Eight common types of geometric transformations: (1) the identity i.e. the original image, (2) a translation, (3) a rotation, (4) a uniform scaling, (5) a perspective projection, (6) a shear, (7) a reflection, and (8) a nonuniform scaling. . . . 5

2.4 The undesired modal nature of the two-finger largest distance approach. (1) Three fingers are laid down on a geometric object so that one is between the other two at the middle of the object. (2) The one is moved away from the middle and seems not to affect the object in any way. (3) If the one is moved far enough, it suddenly starts to affect the object by stealing the control from either one of the other two. . . . 6

2.5 A N-pointer transformation estimation problem illustrated with 3 points. (Left) We are given domain points $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ and range points $\mathbf{y}_1$, $\mathbf{y}_2$, and $\mathbf{y}_3$. (Right) Our task is to find such a transformation that when the domain points are transformed, the sum of their squared distances ϵ_1^2, ϵ_2^2, and ϵ_3^2 to the range points is minimal. . . . 8

4.1 An example where the loss remains the same regardless of the rotation but neither the domain points $\mathbf{x}_1$, $\mathbf{x}_2$ nor the range points $\mathbf{y}_1$, $\mathbf{y}_2$ are at the pivot $\mathbf{p}$ . . . 26

5.1 Dynamics of the number of concurrent touch points. (1) At first, a user is translating a geometric object with one finger. (2) Then, during the translation, the user applies a small rotation to the object with a second finger. (3) The user lifts the second finger and should be able to continue the translation with the first. . . . 45

5.2 In Taaspace, HTML elements can be translated, scaled, and rotated into nontrivial arrangements thanks to the algorithms presented in this thesis. . . . 48

6.1 Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 10 point pairs. A higher bar is better. 52
6.2 Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 100 point pairs. A higher bar is better. 52
6.3 Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 10,000 point pairs. A higher bar is better. 52
6.4 Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 1,000,000 point pairs. A higher bar is better. 52

1 INTRODUCTION

In this thesis, we present how unlimited number of fingers and users can be equally respected on a touch screen to rotate, scale, and translate geometric objects in a way that is more general, natural, and error-free than current, widely spread two-finger multi-touch transformation gestures. Figure 1.1 gives an initial example of an application using the method we are about to present. We provide the results in a format easily accessible to software developers and show the results to be mathematically optimal and computationally efficient.

In Chapter 2, we discuss the current state of multi-touch transformations and their drawbacks. On these drawbacks, we build a motive to propose equal handling of unlimited fingers and users. We find the proposal be a nontrivial optimal estimation problem that fortunately has known solutions in literature but unfortunately they are not yet applied to multi-touch. We also find the solutions in the literature being represented only in matrix algebra notation, thus making them hard to approach by a software developer.

In Chapter 3, due to the challenges found in Chapter 2, we establish a mathematical framework to apply the solutions to multi-touch. First, we formulate the transformation estimation problem and define a matrix representation for 7 types of geometric transformations that are often needed in user interfaces. These are translation, rotation around a pivot, scaling around a pivot, and their composites. Second, we describe known estimation solutions for finding such transformations that are optimal in a sense that they minimize the sum of squared residuals between two sequences of points, here the touch points of the fingers.

In Chapter 4, we build a transformation estimator for each of the 7 types. For each, first we mathematically derive a closed-form solution for an optimal transformation,

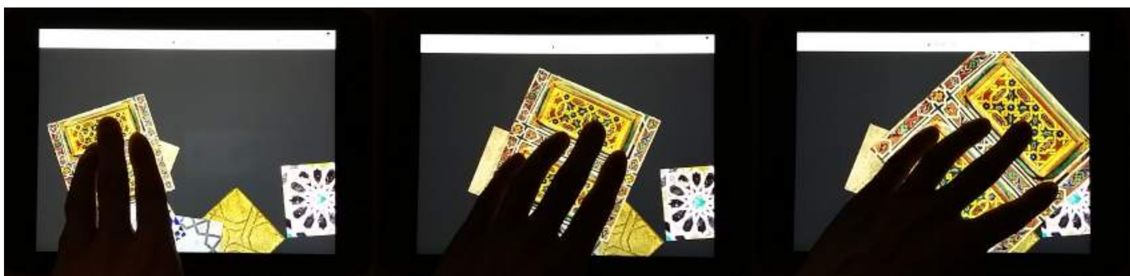

Figure 1.1: *A 4-finger composite rotation, scaling, and translation on a real device, made possible by the results of this thesis.*

then we inspect special cases where the closed-form solution is not defined, and finally and most importantly, represent the results in a consistent, implementation-ready algorithm written in Python for readability.

In Chapter 5, we propose a method how the newly derived algorithms can be applied in the real-world touch user interface programming. We also present a production-ready software package *Nudged* and more experimental package *Taaspace*, which already implement and apply the algorithms to serve needs of web applications. In addition to touch interaction, we show how the algorithms can be applied to spatial correction and, interestingly, geometric layout.

In Chapter 6, we analyze the algorithms from the point of view of their computational efficiency by using computational complexity theoretic concepts *time complexity* and *space complexity*. We favorably find the algorithms to be linear in time and constant in memory consumption. We also measure their actual performance in two web browsers for reference and conclude them to be applicable even on low-end hardware. In addition to efficiency, we discuss initial user feedback and possible improvements such as iteratively weighted least squares to give robustness if input points occasionally contain large measurement errors.

Finally, we conclude the thesis by giving an overview on the results. We also propose topics for subsequent research, namely user experience studies, to further validate the applicability of the results on touch user interfaces.

2 BACKGROUND

To begin the journey, let us first dive into the palpable world of multi-touch interaction and its current challenges. With them in mind, we then become able to propose an improvement and finally, by basing on previous research, describe how it can be made concrete.

2.1 Multi-touch interaction

Since the explosion of mobile devices, touch screens and touch-interactive applications have become more and more apparent in our daily lives. First touch-enabled displays were much like the mouse, allowing only one simultaneous *touch point* [1]. As the displays began to understand multiple touch points, the benefits of multi-finger interaction became more clear. As a consequence, various methods were invented for how movements of fingers could be understood by a computer.

A popular one of such methods is a pinch-like gesture called the *pinch zoom*. In the pinch zoom, as illustrated in Figure 2.1, two fingers become associated with two points on a plane, for example a map, a web page, or a photo. As the fingers move, the plane evenly scales to each direction in a way that the two points remain under the fingers as close as possible. In addition to this *uniform scaling*, sometimes also horizontal and vertical *translation* and *rotation* are allowed.

A reason behind popularity of pinch zoom and *direct touch manipulation* in general is how they succesfully mimic the interaction in the real world [2]. Imagine that you have a piece of cardboard on a slick table like in Figure 2.2. With one finger you can press the piece and move it around. Exactly the same is possible with a touch

Figure 2.1: *The pinch zoom is often used on touch devices to scale geographic maps. The map shrinks when the index finger and the thumb are moved closer to each other during a pinch-like gesture.*

Figure 2.2: *A piece of black cardboard on a slick wooden table.*

screen. The familiarity of the interaction makes it easy for users to discover, learn, and remember the required interaction methods [3].

Another but related benefit of multiple touch points is that they allow the use of multiple hands and the cooperation of multiple users. Especially when a touch surface is large, the use of both hands is a natural method of interaction and there is enough room for several users to benefit from it, in a way similar to the traditional whiteboards used in education and planning. In spite of being still relatively expensive, large touch screens are becoming more and more common. Therefore, robust methods to interact with them are more and more important.

2.2 Geometric transformations

In pinch zoom, a touch screen first captures the finger movements and then a software interprets the movements to scale a geometric object represented on the screen. Scaling, as well as rotation and translation, are *geometric transformations*. Therefore on an abstract level, we are discussing about transformations that are first detected from movements of touch points and then applied to a geometric object.

There are multiple types of geometric transformations and their applications are very common in computer graphics. For example, *projective transformations* are vigorously used in 3D graphics to create a sense of depth. In vector graphics, *affine transformations* such as rotation, scaling, translation, shear, and reflection are common. See Figure 2.3 for visual description of the types.

For the needs of direct touch manipulation, an interesting subset of affine transformations is called the set of *nonreflective similarity transformations* [4] or alternatively *rigid-body transformations* [5]. We choose to use the former due to its clarity because the latter is sometimes used synonymously with *rigid transformations* which

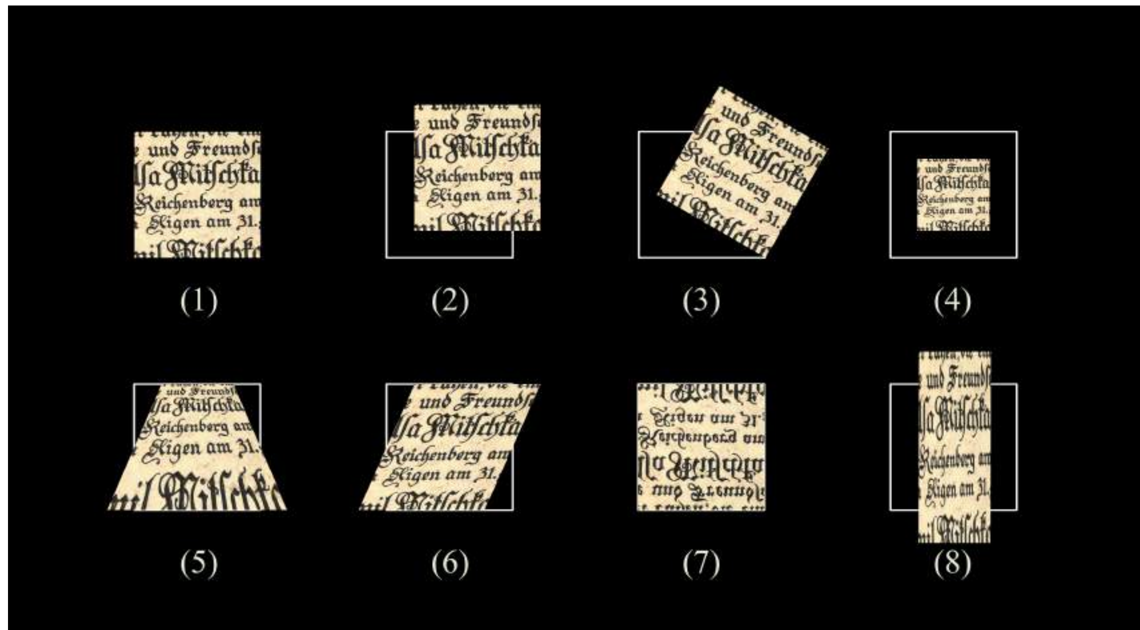

Figure 2.3: *Eight common types of geometric transformations: (1) the identity i.e. the original image, (2) a translation, (3) a rotation, (4) a uniform scaling, (5) a perspective projection, (6) a shear, (7) a reflection, and (8) a nonuniform scaling.*

prohibits the scaling [6]. Anyhow, what makes the nonreflective similarity transformations interesting is how they are ubiquitous in our physical world. They include translation, uniform scaling, and rotation but do not allow shearing, non-uniform scaling, and, as the name tells, reflection. In nature, we cannot stretch or shear a piece of wood or a rock but we still can rotate them and move them closer and farther. Although we can perceive reflection of an object from still water, in our physical world we cannot manipulate an object directly to make it a mirror of the original. Therefore, nonreflective similarity transformations offer us a very familiar and thus powerful interaction space.

2.3 Multi-touch transforming

Rotation, scaling, and translation are common in touch applications. Map applications where the map surface can be moved around, such as *Google Maps* for smartphones, provide a good example. On web browsers, scrolling and zooming (vertical translation and scaling) are extremely common. In games, game characters, inventory items, and other objects are being dragged (translated) around.

However, typically the applications take into account only up to two fingers to compute the transformation. Also, the ways in which the other possible touching fingers are handled differ from application to application. The restriction as well as the heterogeneity lead to multiple inconveniences in the interaction.

A popular multi-touch software package *Hammer.js* is one example [7]. In Hammer.js, the two fingers that define the transformation are the two that first touch

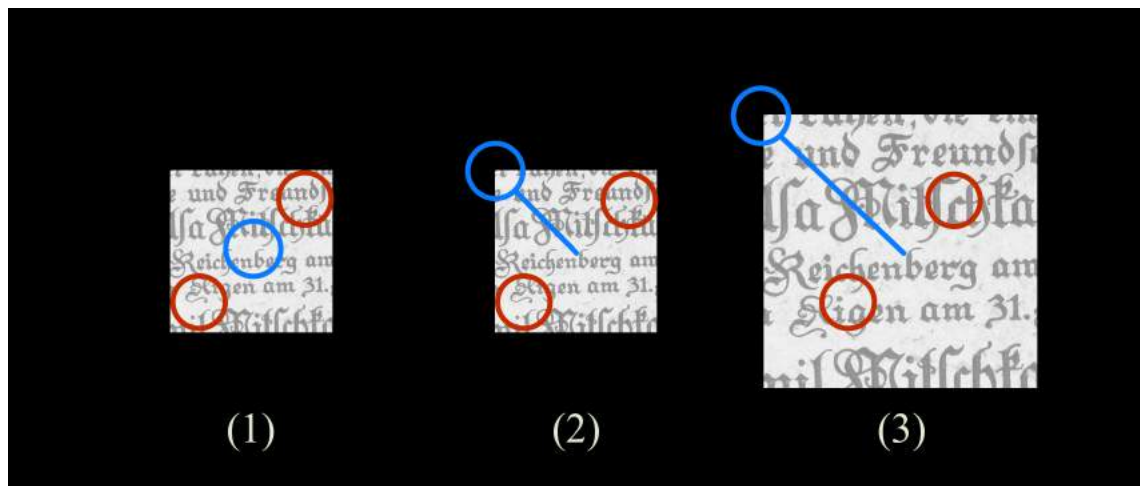

Figure 2.4: *The undesired modal nature of the two-finger largest distance approach. (1) Three fingers are laid down on a geometric object so that one is between the other two at the middle of the object. (2) The one is moved away from the middle and seems not to affect the object in any way. (3) If the one is moved far enough, it suddenly starts to affect the object by stealing the control from either one of the other two.*

the screen. Additional fingers are totally ignored, except in translation where they are equally respected. However, if then one of the fingers is lifted and put back, a sudden unexpected rotation and scaling takes place [8].

The default *Maps* application of Apple mobile operating system iOS 9.3.1 uses similar approach but without unexpected rotation or scaling. However, if a third finger is added and either one of the first two is removed, rotation and scaling are not anymore possible with the remaining ones. In Google Maps version 4.18.0 for Apple iOS, also translation respects only the first two fingers. For additional confusion, two-finger translation *tilts* the map by default, making it look more three-dimensional, but translates it instead if done directly after a rotate or scale.

Respecting only the first fingers can be inefficient and prone to error [9, p. 254]. For example, if a pinky and a ring finger are the first two, the user is able to scale relatively little when compared to scaling with a pinky and a thumb. On the contrary, if the pinky and the ring finger were very close to each other at first, a small gesture could yield a much larger impact than intended.

Probably the best two-finger approach is where the two are the two with the largest distance between them. The captured transformation highly correlates with the gesture [9, p. 254] and works even with two hands because there is probably more distance between the hands than between fingers. Of course, the other fingers must be gracefully handled also in this approach. However, even this has its own quirks. With more than two fingers, it is not always clear which two are respected. Also, users might experience discontinuity in the interaction. For example, as illustrated in Figure 2.4, a third finger between the outermost two does not initially affect the gesture. However, if the third is moved so that it becomes a new outermost, suddenly it starts to affect.

All in all, even though all of these approaches work in typical use cases, they only partially resemble the interaction we are familiar with in the physical world. Anything we touch, the force transmitted to object via our fingertips does neither depend on their order nor their distance relations. A desire to mimic the real-world interacion is not only common sense but supported by results in usability research [2]. For example, the usability expert Jacob Nielsen's 10 design heuristics recommend *"match between system and the real world"* [3]. The same has been recommended in both Microsoft's and Apple's touch design guidelines [10][11].

Additionally, Microsoft's guidelines state: *"Don't use the number of fingers used to distinguish the manipulation whenever possible."* Where the two-finger approaches do not precisely follow this guideline, many operating systems (OSs) directly violate it. Apple iOS 9 by default uses 4 and 5 finger gestures for switching between applications. Where iOS provides a setting to disable this feature, Linux based Ubuntu 14.04 reserves 3 finger gestures for the same purpose but does not provide any easy method to disable them [12]. As a consequence, users are forced to keep the other fingers off the screen, which is burdensome and prone to error.

From this, and previously mentioned challenges, we can hypothesize that users might have learned to avoid using more than two fingers concurrently. The trend might feed itself because users might appear to not need additional fingers. Also, due to the limitations artificially placed by OSs, the application are designed to work only with up to two fingers and conversely the restrictions survive as applications do not ask for support for additional fingers. Therefore in this thesis, it is our job to break the loop by developing an improved approach that avoids the pitfalls and provides distinctive and natural touch experience.

2.4 N-pointer approach

To overcome the problems of the two-finger approaches, in this thesis we propose that by equally taking into account each touch point, we can provide a more natural and predictable user experience. The improved approach should handle an unlimited number of concurrent touch points equally and resemble a real physical interaction. Because not all fingers touch the surface at the same time and new touch points can appear and some disappear during the transformation gesture, the approach is also required to handle the dynamics in the number of touch points.

A key challenge in this approach is how to compute the transformation from the movements of touch points. We can describe the problem as follows (See also Figure 2.5). Given two sequences $\mathcal{X}$ and $\mathcal{Y}$ of two-dimensional points, find a nonreflective similarity transformation that optimally models a mapping from $\mathcal{X}$ to $\mathcal{Y}$. By optimal, we mean that the transformation should map the Nth point of $\mathcal{X}$ as close to the Nth point of $\mathcal{Y}$ as possible, in a way that the sum of their squared distances is minimized. The sequences are obviously required to have the same number of points. How should we solve this mathematical problem?

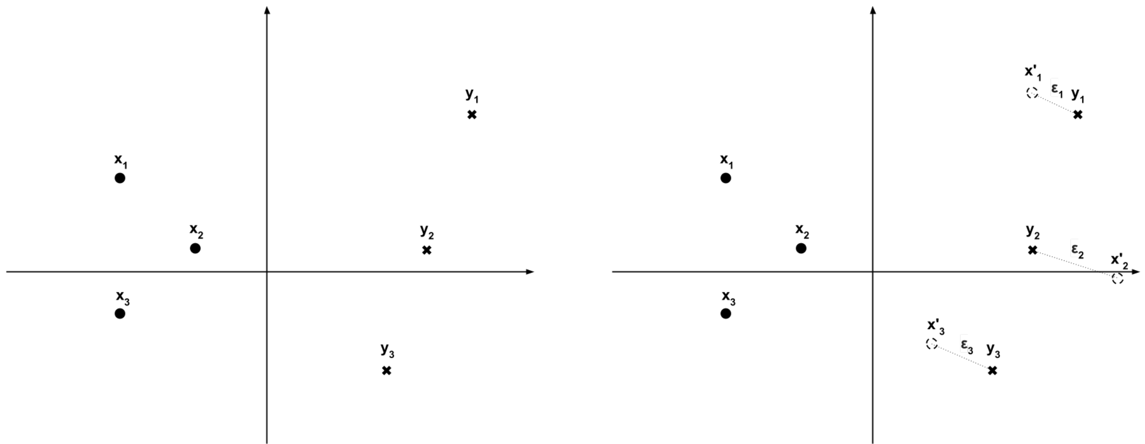

Figure 2.5: *A N-pointer transformation estimation problem illustrated with 3 points. (Left) We are given domain points* $\mathbf{x}_1$*,* $\mathbf{x}_2$*, and* $\mathbf{x}_3$ *and range points* $\mathbf{y}_1$*,* $\mathbf{y}_2$*, and* $\mathbf{y}_3$*. (Right) Our task is to find such a transformation that when the domain points are transformed, the sum of their squared distances* ϵ_1^2*,* ϵ_2^2*, and* ϵ_3^2 *to the range points is minimal.*

Fortunately, a similar problem has been confronted in computer vision, computer graphics, and calibration and called *point set registration*, *point matching*, or *Procrustes superimposition* in literature. The problem has known solutions. For over three decades, multitude of authors have worked on this exact problem and come up with well-defined mathematical methods and solutions. The authors include Kabsch [13], Haralick et al. [14], Maintz & Viergever [15], Umeyama [16], Challis [17], and Alexander et al. [18]. The problem is so common that in 2013 the *Image Processing Toolbox* by MathWorks introduced a function *fitgeotrans* that finds a given type of geometric transformation between given point pairs [4]. The available types include the nonreflective similarity.

Despite the number of publications and delicate methods, none represented the results in a way that would be accessible to an average user interface developer. We cannot assume each developer to have a deep understanding on matrix algebra concepts such as matrix multiplication and inversion, not to mention pseudo-inverses or singular value decomposition. To apply the mathematical results from publications, a developer is required to understand how the matrices behave and what do they represent. To write error-free, quality code, the develop should be aware of special cases that might arise in the solutions, such as singular, non-invertible matrices.

None of the mentioned publications and implementations discuss multi-touch interaction, further raising the barrier for developers. The meaning of mathematically represented solutions and their special cases would need to be interpreted into concepts of touch interaction. To raise the barrier even higher, the publications provide solutions for only a few of the transformation types we are interested in. The developer would thus need to understand also the methods how the solutions were derived and reapply them to obtain the full set of solutions. The readily available fitgeotrans does not help much because it is proprietary and only available for MATLAB, which is not among the programming languages the touch user interfaces are typically build with. Also, for our case, it would provide a solution only for the composite translation, scaling, and rotation.

From all these problems in implementations and shortcomings in literature, no wonder why multi-touch transformations are commonly limited to two fingers. It seems that the designers of these implementations did not have publicly available, understandable, or purpose-fitting reference material to rely on. Also, even if a reference was known, the bar was probably too high and benefits too unknown to reject two-finger approaches. Furthermore, the task of extracting transformation from a two-finger gesture can be relatively easy and thus even attracting for a skilled developer, lowering the interest to invest time to study complex material.

It seems that the world is missing a important reference on how to implement multi-touch transformations for unlimited number of fingers. In this thesis our main goal is to change that. Therefore we need to present an approach where each finger is treated equally without an upper limit for their number. The approach must be capable to handle change in the number of fingers and cover all the important transformation types needed in touch user interfaces. It also needs to do all this in a way we are used to in the physical world.

To ensure the approach can reach developers and eventually users, we represent the set of algorithms in source code listings that are easy to read and implement. The algorithms need to be production-ready, meaning that special cases are taken into account and the algorithms shown to be computationally efficient. This ensures fail-free, fast, and scalable computation and minimizes the need for developers to dive into the mathematical notation.

3 METHOD

In this chapter we describe the challenge of transformation estimation in mathematical terms and do groundwork required to derive estimation algorithms. The mathematical notation used here mostly follows *Elementary Linear Algebra: A Matrix Approach* by Spence et al. [19]. Matrices are denoted with uppercase letters $\boldsymbol{B}$, scalars with lowercase letters b, vectors with bold lowercase letters $\mathbf{b}$, and sets with calligraphic uppercase letters $\mathcal{B}$.

3.1 Problem formulation

Our problem is to find a best estimate for a nonreflective similarity transformation between two point sequences. Let us begin by defining the transformations we are dealing with.

A nonreflective similarity transformation of a two-dimensional real column vector $\mathbf{x}$ to a vector $\mathbf{y}$ can be expressed as a combination of a 2x2 scaling matrix $\boldsymbol{S}$, a 2x2 rotation matrix $\boldsymbol{R}$, and a displacement vector $\mathbf{t}$:

$$\boldsymbol{S}\boldsymbol{R}\mathbf{x} + \mathbf{t} = \mathbf{y} \tag{3.1}$$

Its expanded form is given:

$$\begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{3.2}$$

However, as Möbius presented in 1827 [20], if vectors are represented in *homogeneous coordinates* we can write Equation 3.2 without addition:

$$\begin{bmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} \tag{3.3}$$

We can compose the matrices into one:

$$\begin{bmatrix} \lambda\cos\theta & -\lambda\sin\theta & t_1 \\ \lambda\sin\theta & \lambda\cos\theta & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} \tag{3.4}$$

By letting $s = \lambda\cos\theta$ and $r = \lambda\sin\theta$, we can represent the equation in an alternative, slightly simpler way:

$$\begin{bmatrix} s & -r & t_1 \\ r & s & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} \tag{3.5}$$

The equation can also be compactly written:

$$F\mathbf{x} = \mathbf{y} \tag{3.6}$$

Given only one pair of vectors $(\mathbf{x}, \mathbf{y})$ we have two equations and four unknowns λ, θ, t_1, and t_2. Thus, with only 1 vector pair our system is under-determined and it can have multiple solutions. With 2 pairs the number of equations matches the number of unknowns and therefore an exact solution can be found analytically. We would stop here if we were only interested in two-finger transformations.

However, we are interested in the general case where there is n pairs of vectors. Let us represent them as two matrices $X = \begin{bmatrix} \mathbf{x_1} & \mathbf{x_2} & \cdots & \mathbf{x_n} \end{bmatrix}$ and $Y = \begin{bmatrix} \mathbf{y_1} & \mathbf{y_2} & \cdots & \mathbf{y_n} \end{bmatrix}$. Note that $X, Y \in \mathbb{R}^{3\times n}$. We call X the *domain* and Y the *range*. We write our system simply:

$$FX = Y \tag{3.7}$$

If $n > 2$ then our system has $2n$ equations and thus is over-determined. It does not have solution except in rare cases. Therefore we must incorporate a residual matrix E to even out differences:

$$FX = Y + E \tag{3.8}$$

where $E = \begin{bmatrix} \boldsymbol{\epsilon}_1 \cdots \boldsymbol{\epsilon}_n \end{bmatrix} \in \mathbb{R}^{3\times n}$.

Equation 3.8 formalizes our problem: given X and Y, find F that minimizes the residual E. However, E is a matrix and there could be multiple *loss functions* to measure its minimality, for example the sum of absolute values.

We make a popular decision that the value of the loss function $L(E)$ equals the sum of squared Euclidean distances of residual vectors $\boldsymbol{\epsilon}_1 \cdots \boldsymbol{\epsilon}_n$:

$$L(E) = \sum_{i=1}^{n} \sum_{j=1}^{3} \epsilon_{ij}^2 = \sum_{i=1}^{n} \boldsymbol{\epsilon}_i{}^\top \boldsymbol{\epsilon}_i \tag{3.9}$$

Therefore for $FX = Y + E$, we are interested to find such $\hat{F}$ that minimizes the loss:

$$\begin{aligned} \hat{F} &= \underset{F}{\operatorname{argmin}}\, L(E) \\ &= \underset{F}{\operatorname{argmin}}\, L(FX - Y) \end{aligned} \tag{3.10}$$

Finding $\hat{F}$ is a well-known and solved problem in linear regression. Before we look into that, let us define possible restrictions we would place on $\hat{F}$ in multi-touch interaction.

3.2 Transformation groups

Here we define the different types of nonreflective similarity transformations in group theoretic terms. We will find that in the context of this thesis, it is more convenient to discuss about more vague *types* than exact algebraic groups. However in this section, groups are used. Due to the context of transformations, we especially refer to groups closed under matrix multiplication.

3.2.1 Basic groups

Let us define 3 basic groups of transformations, translations $\mathcal{G}_T$ and uniform scalings $\mathcal{G}_S$ and rotations $\mathcal{G}_R$ around *origin*. Given arbitrary parameters $\lambda, \theta, t_1, t_2 \in \mathbb{R}$, their members $T \in \mathcal{G}_T$, $S \in \mathcal{G}_S$, and $R \in \mathcal{G}_R$ can be defined as follows:

$$T = \begin{bmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

$$S = \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

By composing transformations from these 3 basic groups, we are able to derive more complex groups. As each of the basic groups has its independent free parameters, the composites have a higher *degrees of freedom*. In other words the composites are less restricted.

3.2.2 Composite groups

By composing members from the basic groups we can construct linear systems with desired set of free parameters. For example the system $TRX = Y + E$ describes a system where only parallel movement and rotation are allowed. The system with the most (4) degrees of freedom is given:

$$TSRX = Y + E \tag{3.14}$$

where the domain X, the range Y, and the residual E are as previously described in Section 3.1.

The transformations that result from the composition do not usually belong to any of the basic groups. Composed translation and rotation move a sequence of points in a way that often cannot be represented by either alone. Instead, these composites form their own larger groups. We can immediately think of 4 groups: $\mathcal{G}_{TS}$, $\mathcal{G}_{TR}$, $\mathcal{G}_{SR}$, and $\mathcal{G}_{TSR}$. Their members are defined as follows:

$$TS = \begin{bmatrix} \lambda & 0 & t_1 \\ 0 & \lambda & t_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

$$TR = \begin{bmatrix} \cos\theta & -\sin\theta & t_1 \\ \sin\theta & \cos\theta & t_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

$$SR = \begin{bmatrix} \lambda\cos\theta & -\lambda\sin\theta & 0 \\ \lambda\sin\theta & \lambda\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s & -r & 0 \\ r & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

$$TSR = \begin{bmatrix} \lambda\cos\theta & -\lambda\sin\theta & t_1 \\ \lambda\sin\theta & \lambda\cos\theta & t_2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s & -r & t_1 \\ r & s & t_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

However, the power set of the letters T, S, and R surprisingly yields not 7 but 8 combinations:

$$\mathbb{P}(\{T,S,R\}) = \Big\{\{\},\{T\},\{S\},\{R\},\{T,S\},\{T,R\},\{S,R\},\{T,S,R\}\Big\} \tag{3.19}$$

Interestingly the empty set {} is included. In group theoretic terms this is called the *trivial subgroup* and is included in all the others as a special case. For example zero-degree rotation as well as zero translation equals the identity. In our case, this is the transformation group of the identity transformation, $\mathcal{G}_I$.

Even though it might seem useless to consider a system like $IX = Y + E$, it will turn out to be valuable. We will face situations where the estimation lets at least one transformation parameter to be arbitrary. In those situations, no unique optimal solution can be found and a subset of the allowed transformations are equally valid. However, in multi-touch applications, a randomly chosen arbitrary movement would be annoying. Therefore, our best choice is to choose the simplest best alternative that, as we will see, is often the identity and if not identity, then the optimal translation. Also in machine learning, it is often customary to favor the simplest models [21, p. 713] as well as simply stated by the Ockham's razor.

3.2.3 Pivoted groups

The rotation and scaling groups $\mathcal{G}_S$, $\mathcal{G}_R$ contained only transformations around origin. We are however interested of transformations around any given fixed *pivot* point $\mathbf{p}$. In the context of similarity transformations the pivot is also known as a *center of similarity*. The pivot cannot move during transformation so we have a constraint:

$$F\mathbf{p} = \mathbf{p} \tag{3.20}$$

There is an infinite number of pivots and therefore there is an infinite number of pivoted groups. Together they do not form a group because for example two rotations with different pivots cannot necessarily be represented with any single pivoted rotation. In spite of not being true groups, we label the set of pivoted rotations as $\mathcal{G}_{pR}$ and the set of pivoted scalings as $\mathcal{G}_{pS}$. The set of combined scalings and rotations around a single pivot we denote by $\mathcal{G}_{pSR}$. The set $\mathcal{G}_{pR}$ is a subset of $\mathcal{G}_{TR}$, as well as $\mathcal{G}_{pS}$ is a subset of $\mathcal{G}_{TS}$ and $\mathcal{G}_{pSR}$ is a subset of $\mathcal{G}_{TSR}$, all constrained under Equation 3.20.

Thus, in this thesis we are interested of finding optimal transformations within the following sets and groups, which we collectively call *transformation types*:

1. $\mathcal{G}_T$: the group of translations
2. $\mathcal{G}_{pS}$: the set of scalings around a pivot
3. $\mathcal{G}_{pR}$: the set of rotations around a pivot
4. $\mathcal{G}_{TS}$: the group of translations and scalings
5. $\mathcal{G}_{TR}$: the group of translations and rotations
6. $\mathcal{G}_{pSR}$: the set of scalings and rotations around a pivot

7. $\mathcal{G}_{TSR}$: the group of translations, scalings, and rotations

The list does not include *one-dimensional translations* which would form a group when given a *direction vector*. Like pivoted groups, there is an infinite number of direction vectors and thus an infinite number of one-dimensional translation groups. Although these form a transformation type that would be important for touch manipulation, we leave them to be handled in subsequent studies.

For each of the 7 types, we will derive an estimation algorithm. The central step in this estimation is the linear least squares method, which we will discuss next.

3.3 Least squares estimation

The *linear least squares method* is a popular optimization method to solve linear over-determined systems in the form:

$$A\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\epsilon} \tag{3.21}$$

where $\boldsymbol{\beta} \in \mathbb{R}^{m\times 1}$ is unknown, $A \in \mathbb{R}^{k\times m}$ and $\boldsymbol{\alpha} \in \mathbb{R}^{k\times 1}$ are known, and $\boldsymbol{\epsilon} \in \mathbb{R}^{k\times 1}$ is the residual. Equation 3.21 is known to have an optimal solution for the unknown vector $\boldsymbol{\beta}$ if the *Gramian* matrix $A^\top A$ has an inverse [22][23]. The solution minimizes the squared euclidean distance of $\boldsymbol{\epsilon}$ and is given as:

$$\hat{\boldsymbol{\beta}} = (A^\top A)^{-1} A^\top \boldsymbol{\alpha} \tag{3.22}$$

We however cannot apply it directly. By examining Equation 3.8 ($FX = Y + E$) we understand that our unknown transformation parameters are placed in the matrix F among constants, not in a separate vector. In addition to the different placement of unknowns, X and Y are matrices, not vectors. Thus to apply the solution, Equation 3.8 must be derived to the form of Equation 3.21. Fortunately, we are able to do this.

As the number and placing of the unknown variables depend on the transformation type, the derivation will be a bit different for each. To avoid repetitious work, what we can do independently is the *vectorization* of the matrices X, Y, and E. We first redefine them:

$$X = \begin{bmatrix} \mathbf{x_1} & \cdots & \mathbf{x_n} \end{bmatrix} = \begin{bmatrix} a_1 & \cdots & a_n \\ b_1 & \cdots & b_n \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{a}^\top \\ \mathbf{b}^\top \\ \mathbf{1}^\top \end{bmatrix} \in \mathbb{R}^{3\times n} \tag{3.23}$$

$$Y = \begin{bmatrix} \mathbf{y_1} & \cdots & \mathbf{y_n} \end{bmatrix} = \begin{bmatrix} c_1 & \cdots & c_n \\ d_1 & \cdots & d_n \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{c}^\top \\ \mathbf{d}^\top \\ \mathbf{1}^\top \end{bmatrix} \in \mathbb{R}^{3\times n} \tag{3.24}$$

$$E = \begin{bmatrix} \boldsymbol{\epsilon}_1 & \cdots & \boldsymbol{\epsilon}_n \end{bmatrix} = \begin{bmatrix} e_1 & \cdots & e_n \\ f_1 & \cdots & f_n \\ 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{e}^\top \\ \mathbf{f}^\top \\ \mathbf{0}^\top \end{bmatrix} \in \mathbb{R}^{3\times n} \tag{3.25}$$

The vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}$ are column vectors in $\mathbb{R}^3$. Let p_{ij} be the elements of F. Now we are able to rewrite $FX = Y + E$:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} \mathbf{a}^\top \\ \mathbf{b}^\top \\ \mathbf{1}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{c}^\top \\ \mathbf{d}^\top \\ \mathbf{1}^\top \end{bmatrix} + \begin{bmatrix} \mathbf{e}^\top \\ \mathbf{f}^\top \\ \mathbf{0}^\top \end{bmatrix} \tag{3.26}$$

We multiply the left-hand side:

$$\begin{bmatrix} p_{11}\mathbf{a}^\top + p_{12}\mathbf{b}^\top + p_{13}\mathbf{1}^\top \\ p_{21}\mathbf{a}^\top + p_{22}\mathbf{b}^\top + p_{23}\mathbf{1}^\top \\ p_{31}\mathbf{a}^\top + p_{32}\mathbf{b}^\top + p_{33}\mathbf{1}^\top \end{bmatrix} = \begin{bmatrix} \mathbf{c}^\top \\ \mathbf{d}^\top \\ \mathbf{1}^\top \end{bmatrix} + \begin{bmatrix} \mathbf{e}^\top \\ \mathbf{f}^\top \\ \mathbf{0}^\top \end{bmatrix} \tag{3.27}$$

Now we are able to turn these $3 \times n$ matrices to $3n \times 1$ vectors by reshaping and transposing:

$$\begin{bmatrix} p_{11}\mathbf{a} + p_{12}\mathbf{b} + p_{13}\mathbf{1} \\ p_{21}\mathbf{a} + p_{22}\mathbf{b} + p_{23}\mathbf{1} \\ p_{31}\mathbf{a} + p_{32}\mathbf{b} + p_{33}\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \\ \mathbf{1} \end{bmatrix} + \begin{bmatrix} \mathbf{e} \\ \mathbf{f} \\ \mathbf{0} \end{bmatrix} \tag{3.28}$$

Furthermore we already know that the bottom row of our homogeneous transformation matrices always equals to $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$. Thus $p_{31} = 0$, $p_{32} = 0$, and $p_{33} = 1$, allowing us to ignore the last n trivial equations from the system above. Let us also denote the new residual vector with simple $\boldsymbol{\epsilon}$:

$$\begin{bmatrix} p_{11}\mathbf{a} + p_{12}\mathbf{b} + p_{13}\mathbf{1} \\ p_{21}\mathbf{a} + p_{22}\mathbf{b} + p_{23}\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{3.29}$$

We replace p_{ij} with our transformation parameters s, r, t_1, and t_2 and arrive to what we call our *vectorized system*:

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} \\ r\mathbf{a} + s\mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{3.30}$$

Now, when we are about to derive solutions for each transformation type, we can conveniently begin from this vectorized system. It is much closer to $A\boldsymbol{\beta} = \boldsymbol{\alpha}$ and we only need to derive it so that the desired set of parameters form a vector $\boldsymbol{\beta}$.

The vectorization does not change the optimal solution. Previously (Equation 3.8) we would liked to minimize the sum $\boldsymbol{\epsilon_1}^\top \boldsymbol{\epsilon_1} + \boldsymbol{\epsilon_2}^\top \boldsymbol{\epsilon_2} + \cdots + \boldsymbol{\epsilon_n}^\top \boldsymbol{\epsilon_n} = e_1^2 + f_1^2 + e_2^2 + f_2^2 + \cdots + e_n^2 + f_n^2$. Now, we would like to minimize $\boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} = e_1^2 + \cdots + e_n^2 + f_1^2 + \cdots + f_n^2$. As we can see, the expressions contain the same terms. Therefore they are equal and applying the solution of Equation 3.22 to Equation 3.30 gives us correct estimates.

4 ALGORITHMS AND DERIVATION

In this chapter, we derive 7 optimal estimation algorithms, one for each transformation type listed in Section 3.2.3. We begin with the most restricted ones and continue towards the type with the most free variables, namely $\mathcal{G}_{TSR}$, the unbounded translation, rotation, and uniform scaling.

For each type, we begin from the vectorized linear system of Equation 3.30. By considering the restrictions of each type, we derive a mathematical closed-form solution for estimates of their free parameters. When confronted, we examine the special cases where a unique best solution cannot be found and discuss how these should be interpreted to still result with a meaningful transformation.

For each type, we also compact the closed-form solution and the handling of special cases into an algorithm written in Python. For the needs of the computational efficiency analysis in Chapter 6, we make note of the number of operations and the occurrences of computationally expensive functions. We begin with the translation.

4.1 Translation estimation

When only a translation is allowed, the linear system of Equation 3.8 becomes:

$$\begin{bmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{bmatrix} X = Y + E \tag{4.1}$$

We note that $\lambda = 1$ and $\theta = 0$. Represented in the vectorized form of Equation 3.30 the system thus simplifies to:

$$\begin{bmatrix} \mathbf{a} + t_1\mathbf{1} \\ \mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.2}$$

We move the known $\mathbf{a}$ and $\mathbf{b}$ to right-hand side and then decompose the left-hand side. We arrive to the desired form of $A\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\epsilon}$:

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \mathbf{c} - \mathbf{a} \\ \mathbf{d} - \mathbf{b} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.3}$$

To find $\hat{\boldsymbol{\beta}} = (A^\top A)^{-1} A^\top \boldsymbol{\alpha}$, we first rewrite $A^\top A$ and $(A^\top A)^{-1}$:

$$A^\top A = \begin{bmatrix} \mathbf{1}^\top \mathbf{1} & 0 \\ 0 & \mathbf{1}^\top \mathbf{1} \end{bmatrix} = \begin{bmatrix} n & 0 \\ 0 & n \end{bmatrix} = nI \tag{4.4}$$

$$(A^\top A)^{-1} = n^{-1} I \tag{4.5}$$

The inverse exists only if $n > 0$. If true, then:

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= (A^\top A)^{-1} A^\top \boldsymbol{\alpha} \\ &= n^{-1} \begin{bmatrix} \mathbf{1}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{1}^\top \end{bmatrix} \begin{bmatrix} \mathbf{c} - \mathbf{a} \\ \mathbf{d} - \mathbf{b} \end{bmatrix} \\ &= n^{-1} \begin{bmatrix} \mathbf{1}^\top \mathbf{c} - \mathbf{1}^\top \mathbf{a} \\ \mathbf{1}^\top \mathbf{d} - \mathbf{1}^\top \mathbf{b} \end{bmatrix} \end{aligned} \tag{4.6}$$

which can be written alternatively using summation:

$$\hat{t}_1 = \frac{1}{n} \sum_{i=1}^{n} c_i - \frac{1}{n} \sum_{i=1}^{n} a_i \tag{4.7}$$

$$\hat{t}_2 = \frac{1}{n} \sum_{i=1}^{n} d_i - \frac{1}{n} \sum_{i=1}^{n} b_i \tag{4.8}$$

The interpretation of the result is obvious. If only translation is allowed, the optimal transformation translates the domain mean to match the range mean.

4.1.1 Special cases

Equations 4.7 and 4.8 were defined only if $n > 0$. If $n = 0$ it means that the domain and the range are empty. Without no data, any transformation would be valid but as discussed in Section 3.3, our best guess is the identity. Thus, if $n = 0$ then let $\hat{t}_1 = 0$ and $\hat{t}_2 = 0$.

4.1.2 Algorithm

An algorithm that estimates an optimal translation is given in Listing 4.1. The algorithm takes in X and Y as two *lists* where each element is a two-dimensional point represented as a two-element list $[x_1, x_2]$. The terms $\mathbf{a}^\top \mathbf{1} \ldots \mathbf{d}^\top \mathbf{1}$ are computed as the lists are iterated over. If the length of the lists differ, additional elements in the longer list are ignored. The existence of an unique solution is tested and if true, estimates are computed and returned. If false, estimates representing the identity transformation are returned. The returned 4-tuple defines the matrix in Equation 3.5.

Listing 4.1: *Given a list of domain points* X *and a list of range points* Y *and when only translation is allowed, this function returns optimal similarity transformation matrix parameters* s, r, t1, *and* t2. *The code is written in Python.*

```
def estimate_translation(X, Y):

    N = min(len(X), len(Y))

    a1 = b1 = c1 = d1 = 0

    for i in range(1, N):
        a = X[i][0]
        b = X[i][1]
        c = Y[i][0]
        d = Y[i][1]
        a1 += a
        b1 += b
        c1 += c
        d1 += d

    if N < 1:
        return 1, 0, 0, 0

    s = 1
    r = 0
    t1 = (c1 - a1) / N
    t2 = (d1 - b1) / N

    return s, r, t1, t2
```

To analyze computational demand of the algorithm, a single call with n point pairs requires $4n + 4$ floating-point operations and uses only the basic arithmetic operations. A single call uses a constant amount of memory regardless of n. These are further inspected in Chapter 6.

4.2 Scaling estimation

Next, we derive a method to find an optimal transformation matrix when only scaling around a pivot is allowed. With this type of transformation, $s = \lambda$ and $r = 0$. We define the pivot as:

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \tag{4.9}$$

As discussed in Section 3.2.3, the pivot cannot be moved which gives us a constraint:

$$\begin{bmatrix} s & 0 & t_1 \\ 0 & s & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \tag{4.10}$$

From the constraint we can solve the translation parameters:

$$\begin{aligned} t_1 &= (1-s)p_1 \\ t_2 &= (1-s)p_2 \end{aligned} \tag{4.11}$$

We place the variables into the vectorized system of Equation 3.30:

$$\begin{bmatrix} s\mathbf{a} + ((1-s)p_1)\mathbf{1} \\ s\mathbf{b} + ((1-s)p_2)\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.12}$$

We rearrange the terms:

$$\begin{bmatrix} s(\mathbf{a} - p_1\mathbf{1}) \\ s(\mathbf{b} - p_2\mathbf{1}) \end{bmatrix} = \begin{bmatrix} \mathbf{c} - p_1\mathbf{1} \\ \mathbf{d} - p_2\mathbf{1} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.13}$$

and end up with the desired form $A\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\epsilon}$:

$$\begin{bmatrix} \mathbf{a} - p_1\mathbf{1} \\ \mathbf{b} - p_2\mathbf{1} \end{bmatrix} \begin{bmatrix} s \end{bmatrix} = \begin{bmatrix} \mathbf{c} - p_1\mathbf{1} \\ \mathbf{d} - p_2\mathbf{1} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.14}$$

To simplify notation, let us define *primed* domain and range vectors:

$$\begin{aligned} \mathbf{a}' &= \mathbf{a} - p_1\mathbf{1} \\ \mathbf{b}' &= \mathbf{b} - p_2\mathbf{1} \\ \mathbf{c}' &= \mathbf{c} - p_1\mathbf{1} \\ \mathbf{d}' &= \mathbf{d} - p_2\mathbf{1} \end{aligned} \tag{4.15}$$

We compute the Gramian, its inverse, and $A^\top\boldsymbol{\alpha}$:

$$A^\top A = \begin{bmatrix} \mathbf{a}'^\top & \mathbf{b}'^\top \end{bmatrix} \begin{bmatrix} \mathbf{a}' \\ \mathbf{b}' \end{bmatrix} = \mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}' \tag{4.16}$$

$$(A^\top A)^{-1} = \frac{1}{\mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}'} \tag{4.17}$$

$$\begin{bmatrix}\mathbf{a}'^\top & \mathbf{b}'^\top\end{bmatrix}\begin{bmatrix}\mathbf{c}' \\ \mathbf{d}'\end{bmatrix} = \mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}' \tag{4.18}$$

We arrive to the estimate:

$$\hat{s} = (A^\top A)^{-1}A^\top\boldsymbol{\alpha} = \frac{\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}'}{\mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}'} \tag{4.19}$$

Therefore to estimate an optimal scaling around $\mathbf{p}$, our estimates are:

$$\begin{aligned}\hat{s} &= \frac{\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}'}{\mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}'} \\ \hat{r} &= 0 \\ \hat{t}_1 &= (1-s)p_1 \\ \hat{t}_2 &= (1-s)p_2\end{aligned} \tag{4.20}$$

However, they are not defined if the Gramian is singular. Next, we inspect this special case further.

4.2.1 Special cases

If the Gramian is singular then $\mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}' = 0$. Let us rewrite it with summation:

$$\sum_{i=1}^{n} a'^2_i + \sum_{i=1}^{n} b'^2_i = 0 \tag{4.21}$$

We can now see that all the terms must be positive and thus the Gramian is singular if and only if all the elements are zero. It is equivalent to all the domain points being strictly on $\mathbf{p}$ because $a'_i = a_i - p_1$ and $b'_i = b_i - p_2$.

Because all the domain points are on $\mathbf{p}$, all scalings yield equal loss. Therefore, as λ would be arbitrary, our best guess is $\lambda = 1$ which in this case leads to the identity transformation.

4.2.2 Algorithm

An algorithm that estimates an optimal scaling around a pivot is given in Listing 4.2. The structure of the algorithm is identical to the translation estimation in Section 4.1.2 with the difference given by the pivot. Also, the test for unique solution tests g instead of N. Because g is a floating-point number, its representation for 0 could, due to a floating point rounding error, be not exactly 0 but a very small number and in this case, a small positive number. Therefore in practice when testing for its equality to zero, it is customary to instead test it to be smaller than a small constant *epsilon*.

Listing 4.2: *Given a list of domain points* X*, a list of range points* Y*, and a pivot point* p*, and when only a scaling is allowed, this function returns optimal pivotd scaling matrix parameters* s, r, t1*, and* t2*. The code is written in Python.*

```
def estimate_scaling(X, Y, p):

    N = min(len(X), len(Y))

    ac = 0
    bd = 0
    aa = 0
    bb = 0
    for i in range(1, N):
        a = X[i][0]
        b = X[i][1]
        c = Y[i][0]
        d = Y[i][1]
        ac += a * c
        bd += b * d
        aa += a * a
        bb += b * b

    g = aa + bb

    if g < epsilon:
        return 1, 0, 0, 0

    s = (ac + bd) / g
    r = 0
    t1 = (1 - s) * p[0]
    t2 = (1 - s) * p[1]

    return s, r, t1, t2
```

A single call with n point pairs requires $8n+7$ floating-point operations and uses only the basic arithmetic operations. A single call uses a constant amount of memory regardless of n. These are further inspected in Chapter 6.

4.3 Rotation estimation

In rotation estimation the scaling factor $\lambda = 1$ and therefore $s = \cos\theta$ and $r = \sin\theta$. We try to find an optimal rotation θ around a pivot $\mathbf{p}$. Therefore, we have the following two constraints:

$$\begin{bmatrix} s & -r & t_1 \\ r & s & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \tag{4.22}$$

$$s^2 + r^2 = 1 \tag{4.23}$$

The second constraint is not linear and thus the linear least squares method cannot be applied. Instead, we solve the least squares in more raw method by finding the stationary points of our loss function L. In other words, we find points where the derivative of the summed squared residuals is zero.

We again begin from the vectorized system of Equation 3.30:

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} \\ r\mathbf{a} + s\mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.24}$$

The first constraint allows us to rewrite t_1 and t_2:

$$\begin{aligned} t_1 &= p_1 - p_1 s + p_2 r \\ t_2 &= p_2 - p_1 r - p_2 s \end{aligned} \tag{4.25}$$

Equation 4.24 becomes:

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + (p_1 - p_1 s + p_2 r)\mathbf{1} \\ r\mathbf{a} + s\mathbf{b} + (p_2 - p_1 r - p_2 s)\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.26}$$

We rearrange the terms and now are able to represent the residual vector $\boldsymbol{\epsilon}$ as:

$$\boldsymbol{\epsilon} = \begin{bmatrix} (\mathbf{a} - p_1\mathbf{1})s - (\mathbf{b} - p_2\mathbf{1})r - (\mathbf{c} - p_1\mathbf{1}) \\ (\mathbf{a} - p_1\mathbf{1})r + (\mathbf{b} - p_2\mathbf{1})s - (\mathbf{d} - p_2\mathbf{1}) \end{bmatrix} \tag{4.27}$$

To simplify notation, let us again define the primed domain and range vectors:

$$\begin{aligned} \mathbf{a}' &= \mathbf{a} - p_1\mathbf{1} \\ \mathbf{b}' &= \mathbf{b} - p_2\mathbf{1} \\ \mathbf{c}' &= \mathbf{c} - p_1\mathbf{1} \\ \mathbf{d}' &= \mathbf{d} - p_2\mathbf{1} \end{aligned} \tag{4.28}$$

Now $\boldsymbol{\epsilon}$ is simplified to:

$$\boldsymbol{\epsilon} = \begin{bmatrix} s\mathbf{a}' - r\mathbf{b}' - \mathbf{c}' \\ r\mathbf{a}' + s\mathbf{b}' - \mathbf{d}' \end{bmatrix} \tag{4.29}$$

We remember that $s = \cos\theta$ and $r = \sin\theta$. We are looking for θ that minimizes the loss $L(\boldsymbol{\epsilon})$:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, L(\boldsymbol{\epsilon}) \tag{4.30}$$

The loss can be written as a dot product that furthermore can be written as a matrix multiplication:

$$L(\boldsymbol{\epsilon}) = \sum_{i=1}^{2n} \epsilon_i^2 = \boldsymbol{\epsilon} \cdot \boldsymbol{\epsilon} = \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} \tag{4.31}$$

We can split it into two terms:

$$\begin{aligned} \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} =& (s\mathbf{a}' - r\mathbf{b}' - \mathbf{c}')^\top (s\mathbf{a}' - r\mathbf{b}' - \mathbf{c}') \\ &+ (r\mathbf{a}' + s\mathbf{b}' - \mathbf{d}')^\top (r\mathbf{a}' + s\mathbf{b}' - \mathbf{d}') \end{aligned} \tag{4.32}$$

The dot product as well as the matrix multiplication is distributive. Thus, to compute the terms, we can rely on the following polynomial expansion rules:

$$\begin{aligned} (x - y - z)^2 &= x^2 + y^2 + z^2 - 2xy - 2xz + 2yz \\ (x + y - z)^2 &= x^2 + y^2 + z^2 + 2xy - 2xz - 2yz \end{aligned} \tag{4.33}$$

Therefore, by keeping in mind that $s^2 + r^2 = 1$:

$$\begin{aligned} \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} &= s^2\mathbf{a}'^\top\mathbf{a}' + r^2\mathbf{b}'^\top\mathbf{b}' + \mathbf{c}'^\top\mathbf{c}' - 2rs\mathbf{a}'^\top\mathbf{b}' - 2s\mathbf{a}'^\top\mathbf{c}' + 2r\mathbf{b}'^\top\mathbf{c}' \\ &\quad + r^2\mathbf{a}'^\top\mathbf{a}' + s^2\mathbf{b}'^\top\mathbf{b}' + \mathbf{d}'^\top\mathbf{d}' + 2rs\mathbf{a}'^\top\mathbf{b}' - 2r\mathbf{a}'^\top\mathbf{d}' - 2s\mathbf{b}'^\top\mathbf{d}' \\ &= \mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}' + \mathbf{c}'^\top\mathbf{c}' + \mathbf{d}'^\top\mathbf{d}' \\ &\quad - 2s\mathbf{a}'^\top\mathbf{c}' + 2r\mathbf{b}'^\top\mathbf{c}' - 2r\mathbf{a}'^\top\mathbf{d}' - 2s\mathbf{b}'^\top\mathbf{d}' \end{aligned} \tag{4.34}$$

To take derivative with respect to θ, we need the following two lemmas:

$$\begin{aligned} \frac{ds}{d\theta} &= \frac{d\cos\theta}{d\theta} = -\sin\theta = -r \\ \frac{dr}{d\theta} &= \frac{d\sin\theta}{d\theta} = \cos\theta = s \end{aligned} \tag{4.35}$$

We also find that we can neglect the first 4 terms. The derivative becomes:

$$\begin{aligned}\frac{d\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}}{d\theta} &= \frac{d}{d\theta}\Big(- 2s\mathbf{a}'^\top\mathbf{c}' + 2r\mathbf{b}'^\top\mathbf{c}' - 2r\mathbf{a}'^\top\mathbf{d}' - 2s\mathbf{b}'^\top\mathbf{d}'\Big) \\ &= 2r\mathbf{a}'^\top\mathbf{c}' + 2s\mathbf{b}'^\top\mathbf{c}' - 2s\mathbf{a}'^\top\mathbf{d}' + 2r\mathbf{b}'^\top\mathbf{d}'\end{aligned} \tag{4.36}$$

We set the derivative to zero and rearrange the terms.

$$(\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}')r + (\mathbf{b}'^\top\mathbf{c}' - \mathbf{a}'^\top\mathbf{d}')s = 0 \tag{4.37}$$

Therefore we find the optimal solution to occur when:

$$\frac{\mathbf{a}'^\top\mathbf{d}' - \mathbf{b}'^\top\mathbf{c}'}{\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}'} = \frac{r}{s} = \frac{\sin\theta}{\cos\theta} = \tan\theta \tag{4.38}$$

which would give us an optimal $\hat{\theta}$. However, we cannot yet directly represent r without s and vice versa. Fortunately, their trigonometric nature allows us to think them as edges of a triangle. On a right-angled triangle with catheti a, b and hypotenuse γ, let $\tan\theta = a/b$ so that $r = \sin\theta = a/\gamma$ and $s = \cos\theta = b/\gamma$. Now the Pythagorean theorem gives us $r = a/\sqrt{a^2 + b^2}$ and $s = b/\sqrt{a^2 + b^2}$. Hence,

$$\begin{aligned}\hat{r} &= \gamma^{-1}(\mathbf{a}'^\top\mathbf{d}' - \mathbf{b}'^\top\mathbf{c}') \\ \hat{s} &= \gamma^{-1}(\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}') \\ \hat{t_1} &= p_1 - p_1\hat{s} + p_2\hat{r} \\ \hat{t_2} &= p_2 - p_1\hat{r} - p_2\hat{s}\end{aligned} \tag{4.39}$$

where

$$\gamma = \sqrt{(\mathbf{a}'^\top\mathbf{d}' - \mathbf{b}'^\top\mathbf{c}')^2 + (\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}')^2} \tag{4.40}$$

This concludes our task to find an optimal rotation matrix. However, the optimal solution does not exist if $\gamma = 0$. We inspect this special case further.

4.3.1 Special cases

We cannot find a unique solution if $\gamma = 0$. By analyzing Equation 4.40 we see that the root and the squares cannot yield negative value. Therefore $\gamma = 0$ if and only if $\mathbf{a}'^\top\mathbf{d}' - \mathbf{b}'^\top\mathbf{c}' = 0$ and $\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}' = 0$. Because they are zero, Equation 4.37 becomes $0r + 0s = 0$ which is equivalent to $0\sin\theta + 0\cos\theta = 0$. Therefore any θ would satisfy this equation, thus be optimal.

Let us inspect with which kind of point configurations this can happen. For example, if each domain point $\mathbf{x}_i$ is at the pivot $\mathbf{p}$ then all rotations would yield an equal loss;

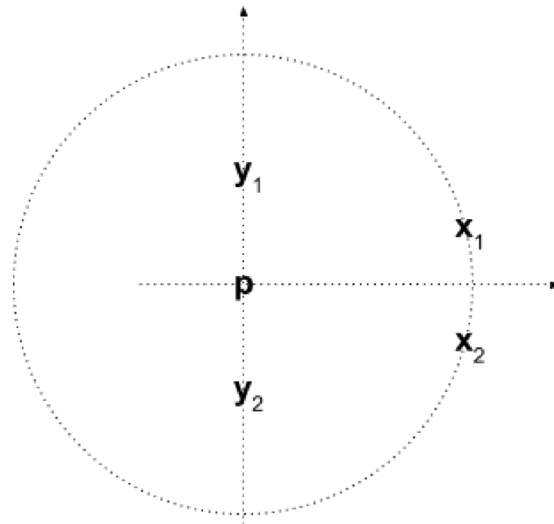

Figure 4.1: *An example where the loss remains the same regardless of the rotation but neither the domain points* $\mathbf{x}_1$, $\mathbf{x}_2$ *nor the range points* $\mathbf{y}_1$, $\mathbf{y}_2$ *are at the pivot* $\mathbf{p}$

no domain point cannot move at all. Likewise, if each range point $\mathbf{y}_i$ lies at $\mathbf{p}$ then no rotation would move the domain points closer or farther from the range. In the former case, the primed domain vectors $\mathbf{a}' = \mathbf{b}' = \mathbf{0}$ because $\mathbf{a}' = \mathbf{a} - p_1\mathbf{1}$ and $\mathbf{b}' = \mathbf{b} - p_2\mathbf{1}$. Identically in the latter case $\mathbf{c}' = \mathbf{d}' = \mathbf{0}$. The both cases lead to $\gamma = 0$.

Another case is illustrated in Figure 4.1. Here none of the points are at pivot but the domain and the range are arranged so that any rotation would yield an equal loss. From this, we can infer that there is an infinite number of these special cases. Fortunately for us, it is enough to know if $\gamma = 0$ and that if true, then all rotations are equally valid. As discussed in Section 3.2.2 we should choose the simplest alternative which here is the identity transform.

4.3.2 Algorithm

An algorithm that estimates an optimal rotation around a pivot is given in Listing 4.3. The structure of the algorithm is identical to the scaling estimation in Section 4.1.2.

Listing 4.3: *Given a list of domain points* X*, a list of range points* Y*, and a pivot point* p*, this function returns optimal pivoted rotation matrix parameters* s, r, t1, *and* t2*. The code is written in Python.*

```
def estimate_rotation(X, Y, p):

    N = min(len(X), len(Y))

    p1 = p[0]
    p2 = p[1]
    ac = 0
    ad = 0
```

```
bc = 0
bd = 0
for i in range(1, N):
    a = X[i][0] - p1
    b = X[i][1] - p2
    c = Y[i][0] - p1
    d = Y[i][1] - p2
    ac += a * c
    ad += a * d
    bc += b * c
    bd += b * d

v = ac + bd
w = ad - bc
g = sqrt(v * v + w * w)

if g < epsilon:
    return 1, 0, 0, 0

s = v / g
r = w / g
t1 = p1 - p1 * s + p2 * r
t2 = p2 - p1 * r - p2 * s;

return s, r, t1, t2
```

A single call with n point pairs executes $12n+16$ floating-point operations, including 1 square root. A single call uses a constant amount of memory regardless of n. These are further inspected in Chapter 6.

4.4 Translation-scaling estimation

Here we derive a method to estimate a nonreflective similarity transformation when rotation is not allowed, thus $s = \lambda$ and $r = 0$. This differs from the pivoted scaling estimation in the absence of the pivot constraint. We again start from the vectorized form of Equation 3.30:

$$\begin{bmatrix} s\mathbf{a} + t_1\mathbf{1} \\ s\mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.41}$$

This is equivalent to the following equation in the form of $A\boldsymbol{\beta} = \boldsymbol{\alpha}$:

$$\begin{bmatrix} \mathbf{a} & \mathbf{1} & \mathbf{0} \\ \mathbf{b} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} s \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.42}$$

We know it to have an optimal solution $\hat{\boldsymbol{\beta}} = (A^\top A)^{-1} A^\top \boldsymbol{\alpha}$ if the Gramian $A^\top A$ has an inverse. Before deriving the Gramian, let us define three variables for notational convenience:

$$\begin{aligned} u &= \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} \\ v &= \mathbf{a}^\top \mathbf{1} = \mathbf{1}^\top \mathbf{a} \\ w &= \mathbf{b}^\top \mathbf{1} = \mathbf{1}^\top \mathbf{b} \end{aligned} \tag{4.43}$$

We derive the Gramian, its inverse, and $A^\top \boldsymbol{\alpha}$. To compute the inverse, we can use the computational knowledge engine *Wolfram Alpha* [24]. The results are as follows:

$$A^\top A = \begin{bmatrix} \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} & \mathbf{a}^\top \mathbf{1} & \mathbf{b}^\top \mathbf{1} \\ \mathbf{1}^\top \mathbf{a} & \mathbf{1}^\top \mathbf{1} & 0 \\ \mathbf{1}^\top \mathbf{b} & 0 & \mathbf{1}^\top \mathbf{1} \end{bmatrix} = \begin{bmatrix} u & v & w \\ v & n & 0 \\ w & 0 & n \end{bmatrix} \tag{4.44}$$

$$(A^\top A)^{-1} = \frac{1}{\gamma} \begin{bmatrix} n^2 & -nv & -nw \\ -nv & nu - w^2 & vw \\ -nw & vw & nu - v^2 \end{bmatrix} \tag{4.45}$$

$$\gamma = n^2 u - n v^2 - n w^2 \tag{4.46}$$

$$A^\top \boldsymbol{\alpha} = \begin{bmatrix} \mathbf{a}^\top & \mathbf{b}^\top \\ \mathbf{1}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{1}^\top \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d} \\ \mathbf{c}^\top \mathbf{1} \\ \mathbf{d}^\top \mathbf{1} \end{bmatrix} \tag{4.47}$$

We arrive to the estimated vector $\hat{\boldsymbol{\beta}}$:

$$(A^\top A)^{-1} A^\top \boldsymbol{\alpha} = \gamma^{-1} \begin{bmatrix} n^2 \mathbf{a}^\top \mathbf{c} + n^2 \mathbf{b}^\top \mathbf{d} - nv \mathbf{c}^\top \mathbf{1} - nw \mathbf{d}^\top \mathbf{1} \\ -nv \mathbf{a}^\top \mathbf{c} - nv \mathbf{b}^\top \mathbf{d} + (nu - w^2) \mathbf{c}^\top \mathbf{1} + vw \mathbf{d}^\top \mathbf{1} \\ -nw \mathbf{a}^\top \mathbf{c} - nw \mathbf{b}^\top \mathbf{d} + vw \mathbf{c}^\top \mathbf{1} + (nu - v^2) \mathbf{d}^\top \mathbf{1} \end{bmatrix} \tag{4.48}$$

We open v and w and write the estimates:

$$\begin{aligned} \hat{s} &= \gamma^{-1}\Big(n^2(\mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d}) - n(\mathbf{a}^\top \mathbf{1})(\mathbf{c}^\top \mathbf{1}) - n(\mathbf{b}^\top \mathbf{1})(\mathbf{d}^\top \mathbf{1})\Big) \\ \hat{r} &= r = 0 \\ \hat{t}_1 &= \gamma^{-1}\Big(- n\mathbf{a}^\top \mathbf{1}\ \mathbf{a}^\top \mathbf{c} - n\mathbf{a}^\top \mathbf{1}\ \mathbf{b}^\top \mathbf{d} + nu\mathbf{c}^\top \mathbf{1} - \mathbf{b}^\top \mathbf{1}\ \mathbf{b}^\top \mathbf{1}\ \mathbf{c}^\top \mathbf{1} + \mathbf{a}^\top \mathbf{1}\ \mathbf{b}^\top \mathbf{1}\ \mathbf{d}^\top \mathbf{1}\Big) \\ \hat{t}_2 &= \gamma^{-1}\Big(- n\mathbf{b}^\top \mathbf{1}\ \mathbf{a}^\top \mathbf{c} - n\mathbf{b}^\top \mathbf{1}\ \mathbf{b}^\top \mathbf{d} + nu\mathbf{d}^\top \mathbf{1} - \mathbf{a}^\top \mathbf{1}\ \mathbf{a}^\top \mathbf{1}\ \mathbf{d}^\top \mathbf{1} + \mathbf{a}^\top \mathbf{1}\ \mathbf{b}^\top \mathbf{1}\ \mathbf{c}^\top \mathbf{1}\Big) \end{aligned} \tag{4.49}$$

where

$$
\begin{aligned}
\gamma &= n^2 u - n(\mathbf{a}^\top \mathbf{1})^2 - n(\mathbf{b}^\top \mathbf{1})^2 \\
u &= \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b}
\end{aligned}
\tag{4.50}
$$

This concludes our task to find an optimal translation-scaling matrix. However, the optimal solution does not exist if $\gamma = 0$. Next, we inspect this special case further.

4.4.1 Special cases

If $\gamma = 0$ then a unique solution cannot be found. In software, to choose best alternative instead of an error, let us first examine when does this happen.

Trivially if $n = 0$ then $\gamma = 0$. No domain points were given so the identity transformation is the simplest choice.

If $n = 1$ then $\gamma = a_1^2 + b_1^2 - a_1^2 - b_1^2 = 0$, which is true for any a_1 and b_1. A single domain and range point were given so any translation-scaling that translates the domain point to range would do. The simplest best choice is then to let the scaling factor be 1 and do only optimal translation described in Section 4.1.

If $n = 2$ then $\gamma = 4(a_1^2 + a_2^2) + 4(b_1^2 + b_2^2) - 2(a_1 + a_2)^2 - 2(b_1 + b_2)^2$. Letting $\gamma = 0$ and expanding the binomials give us $a_1^2 + a_2^2 + b_1^2 + b_2^2 - 2a_1a_2 - 2b_1b_2 = 0$. By reordering and simplifying, we arrive to $(a_1 - a_2)^2 + (b_1 - b_2)^2 = 0$. As the squares of both binomials are non-negative, $\gamma = 0$ if and only if $a_1 = a_2$ and $b_1 = b_2$ i.e. the 2 domain points equal. Next, we proof this to be true for any $n > 0$.

Theorem 4.4.1. *Let $\gamma = n^2\mathbf{a}^\top\mathbf{a} + n^2\mathbf{b}^\top\mathbf{b} - n(\mathbf{a}^\top\mathbf{1})^2 - n(\mathbf{b}^\top\mathbf{1})^2$. Now, $\gamma = 0$ if and only if $a_1 = a_2 = \cdots = a_n$ and $b_1 = b_2 = \cdots = b_n$ for any $n > 0$.*

Proof. We prove this by induction. The previous $n = 1$ case works as our induction basis. For the inductive hypothesis, let us assume that for n known points $\gamma_n = 0$ and that $\gamma_n = 0 \Leftrightarrow \forall i, j \in [1..n] : a_i = a_j \wedge b_i = b_j$. To ease manipulation, let us rewrite $\gamma_n = 0$ with summations:

$$
n\sum_{i=1}^{n} a_i^2 + n\sum_{i=1}^{n} b_i^2 - \Big(\sum_{i=1}^{n} a_i\Big)^2 - \Big(\sum_{i=1}^{n} b_i\Big)^2 = 0 \tag{4.51}
$$

For the inductive step, we add a $(n+1)$th point and show that $\gamma_{n+1} = 0$ if and only if $\forall i, j \in [1..n+1] : a_i = a_j \wedge b_i = b_j$.

$$
(n+1)\sum_{i=1}^{n+1} a_i^2 + (n+1)\sum_{i=1}^{n+1} b_i^2 - \Big(\sum_{i=1}^{n+1} a_i\Big)^2 - \Big(\sum_{i=1}^{n+1} b_i\Big)^2 = 0 \tag{4.52}
$$

We take $(n+1)$th terms out of the summations to find the left-hand side of Equation 4.51:

$$\begin{aligned} n\Big(\sum_{i=1}^{n} a_i^2\Big) + na_{n+1}^2 + \Big(\sum_{i=1}^{n} a_i^2\Big) + a_{n+1}^2 \\ +n\Big(\sum_{i=1}^{n} b_i^2\Big) + nb_{n+1}^2 + \Big(\sum_{i=1}^{n} b_i^2\Big) + b_{n+1}^2 \\ -\Big(\sum_{i=1}^{n} a_i\Big)^2 - 2a_{n+1}\Big(\sum_{i=1}^{n} a_i\Big) - a_{n+1}^2 \\ -\Big(\sum_{i=1}^{n} b_i\Big)^2 - 2b_{n+1}\Big(\sum_{i=1}^{n} b_i\Big) - b_{n+1}^2 = 0 \end{aligned} \tag{4.53}$$

The hypothesis states $\gamma_n = 0$ and therefore Equation 4.53 simplifies to:

$$\begin{aligned} na_{n+1}^2 + \Big(\sum_{i=1}^{n} a_i^2\Big) + nb_{n+1}^2 + \Big(\sum_{i=1}^{n} b_i^2\Big) \\ -2a_{n+1}\Big(\sum_{i=1}^{n} a_i\Big) - 2b_{n+1}\Big(\sum_{i=1}^{n} b_i\Big) = 0 \end{aligned} \tag{4.54}$$

We note that $nx^2 = \sum_{i=1}^{n} x^2$, reorder the terms, and apply $x^2 - 2xy + y^2 = (x-y)^2$:

$$\sum_{i=1}^{n}(a_{n+1} - a_i)^2 + \sum_{i=1}^{n}(b_{n+1} - b_i)^2 = 0 \tag{4.55}$$

This is true if and only if $\forall i, j \in [1..n+1] : a_i = a_j \wedge b_i = b_j$, which completes the proof. □

Therefore if $n > 0$ and $\gamma = 0$, all the domain points are at the same location. No scaling could spread them, so our best choice for optimal transformation is the optimal translation to the range mean as described in Section 4.1.

4.4.2 Algorithm

An algorithm that estimates an optimal composite translation and scaling is given in Listing 4.4. The structure of the algorithm is similar to the translation estimation algorithm in Section 4.1.2. However, handling of special cases is slightly more complicated. If γ, represented by g in the code, is close to zero, either number of points is zero or all domain points are equal. If the former, the identity transform parameters are returned. If the latter, then parameters for an optimal translation to the mean of the range are returned.

Listing 4.4: *Given a list of domain points* X *and a list of range points* Y*, this function returns optimal translation-scaling matrix parameters* s, r, t1*, and* t2*. The code is written in Python*

```
def estimate_translation_scaling(X, Y):

    N = min(len(X), len(Y))

    a1 = b1 = c1 = d1 = 0
    a2 = b2 = ac = bd = 0
    for i in range(1, N):
        a = X[i][0]
        b = X[i][1]
        c = Y[i][0]
        d = Y[i][1]
        a1 += a
        b1 += b
        c1 += c
        d1 += d
        a2 += a * a
        b2 += b * b
        ac += a * c
        bd += b * d

    N2 = N * N
    a12 = a1 * a1
    b12 = b1 * b1
    u = a2 + b2
    v = ac + bd
    g = N2 * u - N * (a12 + b12)

    if g < epsilon:
        if N == 0:
            return 1, 0, 0, 0
        return 1, 0, (c1 / N) - a, (d1 / N) - b

    a1c1 = a1 * c1
    b1d1 = b1 * d1

    s = (N2 * v - N * (a1c1 + b1d1)) / g
    r = 0
    t1 = (N * (c1 * u - a1 * v) - b12 * c1 + a1 * b1d1) / g
    t2 = (N * (d1 * u - b1 * v) - a12 * d1 + b1 * a1c1) / g

    return s, r, t1, t2
```

A single call with n point pairs executes $12n + 34$ basic arithmetic floating-point operations. A single call uses constant amount of memory regardless of n. These are further inspected in Chapter 6.

4.5 Translation-rotation estimation

Here we derive a method to estimate an optimal nonreflective similarity transformation when scaling is not allowed, thus $\lambda = 1$ and furthermore $s = \cos\theta$ and $r = \sin\theta$. As with the pivoted rotation of Section 4.3, the absence of scaling requires $s^2+r^2 = 1$ which makes our linear over-determined system to have non-linear constraints and thus prevents us from using linear least squares solution of Equation 3.22.

As in Section 4.3, we find the least-squares solution through differentiation, although this time our system has 3 unknowns, θ, t_1, and t_2. By forming the loss function and setting its partial derivatives i.e. the *gradient* vector to zero, we can find the minimum of the loss.

We again begin with the vectorized system given in Equation 3.30:

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} \\ r\mathbf{a} + s\mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.56}$$

By rearranging the terms, we easily find the residual:

$$\boldsymbol{\epsilon} = \begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} - \mathbf{c} \\ r\mathbf{a} + s\mathbf{b} + t_2\mathbf{1} - \mathbf{d} \end{bmatrix} \tag{4.57}$$

The loss $\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}$ has a lengthy expression, exceeding the length of the expressions we saw in Section 4.3 and thus we skip it here. It can however be derived by applying the following polynomial expansion rule:

$$(a + b + c + d)^2 = a^2 + 2ab + 2ac + 2ad + b^2 + 2bc + 2bd + c^2 + 2cd + d^2 \tag{4.58}$$

Nevertheless, the partial derivatives of the loss are:

$$\begin{aligned} \frac{\delta}{\delta\theta}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon} = & \; 2r(\mathbf{a}^\top\mathbf{c} + \mathbf{b}^\top\mathbf{d} - t_1\mathbf{a}^\top\mathbf{1} - t_2\mathbf{b}^\top\mathbf{1}) \\ & -2s(\mathbf{a}^\top\mathbf{d} - \mathbf{b}^\top\mathbf{c} - t_2\mathbf{a}^\top\mathbf{1} + t_1\mathbf{b}^\top\mathbf{1}) \end{aligned} \tag{4.59}$$

$$\frac{\delta}{\delta t_1}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon} = 2(s\mathbf{a}^\top\mathbf{1} - r\mathbf{b}^\top\mathbf{1} - \mathbf{c}^\top\mathbf{1} + nt_1) \tag{4.60}$$

$$\frac{\delta}{\delta t_2}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon} = 2(r\mathbf{a}^\top\mathbf{1} - s\mathbf{b}^\top\mathbf{1} - \mathbf{d}^\top\mathbf{1} + nt_2) \tag{4.61}$$

Setting Equation 4.60 and 4.61 to zero and rearranging the terms gives us:

$$\begin{aligned} t_1 &= n^{-1}(-s\mathbf{a}^\top\mathbf{1} + r\mathbf{b}^\top\mathbf{1} + \mathbf{c}^\top\mathbf{1}) \\ t_2 &= n^{-1}(-r\mathbf{a}^\top\mathbf{1} - s\mathbf{b}^\top\mathbf{1} + \mathbf{d}^\top\mathbf{1}) \end{aligned} \tag{4.62}$$

Settings Equation 4.59 to zero, expanding t_1 and t_2, and simplifying the result gives us:

$$\begin{aligned} &r\Big(n\mathbf{a}^\top\mathbf{c} + n\mathbf{b}^\top\mathbf{d} - (\mathbf{a}^\top\mathbf{1})(\mathbf{c}^\top\mathbf{1}) - (\mathbf{b}^\top\mathbf{1})(\mathbf{d}^\top\mathbf{1})\Big) \\ &- s\Big(n\mathbf{a}^\top\mathbf{d} - n\mathbf{b}^\top\mathbf{c} - (\mathbf{a}^\top\mathbf{1})(\mathbf{d}^\top\mathbf{1}) + (\mathbf{b}^\top\mathbf{1})(\mathbf{c}^\top\mathbf{1})\Big) = 0 \end{aligned} \tag{4.63}$$

Let:

$$\begin{aligned} u &= n\mathbf{a}^\top\mathbf{c} + n\mathbf{b}^\top\mathbf{d} - (\mathbf{a}^\top\mathbf{1})(\mathbf{c}^\top\mathbf{1}) - (\mathbf{b}^\top\mathbf{1})(\mathbf{d}^\top\mathbf{1}) \\ v &= n\mathbf{a}^\top\mathbf{d} - n\mathbf{b}^\top\mathbf{c} - (\mathbf{a}^\top\mathbf{1})(\mathbf{d}^\top\mathbf{1}) + (\mathbf{b}^\top\mathbf{1})(\mathbf{c}^\top\mathbf{1}) \end{aligned} \tag{4.64}$$

Now Equation 4.63 becomes a simple:

$$ru - sv = 0 \tag{4.65}$$

By applying Pythagorean theorem like in Section 4.3, we end up with the estimates. First, let:

$$\gamma = \sqrt{u^2 + v^2} \tag{4.66}$$

Then we can write the estimates:

$$\begin{aligned} \hat{s} &= \gamma^{-1}u \\ \hat{r} &= \gamma^{-1}v \\ \hat{t}_1 &= n^{-1}(-\hat{s}\mathbf{a}^\top\mathbf{1} + \hat{r}\mathbf{b}^\top\mathbf{1} + \mathbf{c}^\top\mathbf{1}) \\ \hat{t}_2 &= n^{-1}(-\hat{r}\mathbf{a}^\top\mathbf{1} - \hat{s}\mathbf{b}^\top\mathbf{1} + \mathbf{d}^\top\mathbf{1}) \end{aligned} \tag{4.67}$$

The estimates are not defined if $n = 0$ or $\gamma = 0$. Next, we inspect this special case further.

4.5.1 Special cases

Trivially, if $n = 0$ then the identity is the best choice.

For $n > 0$, if $\gamma = 0$ we cannot find a unique solution. As we saw in Chapter 4.3, $\gamma = 0$ if and only if $u = 0 \wedge v = 0$. By looking Equation 4.65, we can again deduce that if $\gamma = 0$ any θ is equally optimal.

If $\gamma = 0$, let us choose not to rotate at all, i.e. $\theta = 0$. Then $\hat{s} = 1$ and $\hat{r} = 0$, so $\hat{t}_1$ and $\hat{t}_2$ becomes:

$$\hat{t}_1 = \frac{1}{n}\sum_{i=1}^{n} c_i - \frac{1}{n}\sum_{i=1}^{n} a_i \tag{4.68}$$

$$\hat{t}_2 = \frac{1}{n}\sum_{i=1}^{n} d_i - \frac{1}{n}\sum_{i=1}^{n} b_i \tag{4.69}$$

which is equivalent to the translation from the mean of the domain to the mean of the range. Therefore, as previously seen in Section 4.4.1, the optimal translation is our simplest best choice.

4.5.2 Algorithm

An algorithm that estimates an optimal translation-rotation is given in Listing 4.5. The structure and handling of special cases are identical to the translation-scaling algorithm in Section 4.4.2.

Listing 4.5: *Given a list of domain points* X *and a list of range points* Y, *this function returns optimal translation-rotation matrix parameters* s, r, t1, *and* t2. *The code is written in Python*

```
def estimate_translation_rotation(X, Y):

    N = min(len(X), len(Y))

    a1 = b1 = c1 = d1 = 0
    ac = ad = bc = bd = 0
    for i in range(1, N):
        a = X[i][0]
        b = X[i][1]
        c = Y[i][0]
        d = Y[i][1]
        a1 += a
        b1 += b
        c1 += c
        d1 += d
```

```
        ac += a * c
        ad += a * d
        bc += b * c
        bd += b * d

    u = N * (ac + bd) - a1 * c1 - b1 * d1
    v = N * (ad - bc) - a1 * d1 + b1 * c1
    g = sqrt(u * u + v * v)

    if g < epsilon:
        if N == 0:
            return 1, 0, 0, 0
        return 1, 0, (c1 - a1) / N, (d1 - b1) / N

    s = u / g
    r = v / g
    t1 = (-a1 * s + b1 * r + c1) / N
    t2 = (-a1 * r - b1 * s + d1) / N

    return s, r, t1, t2
```

A single call with n point pairs executes $12n + 27$ basic arithmetic floating-point operations and 1 floating-point square root. A single call uses constant amount of memory regardless of n. These are further inspected in Chapter 6.

4.6 Scaling-rotation estimation

Here we derive a method to estimate a nonreflective similarity transformation when both scaling and rotation are allowed around a pivot. We now have the full set of free variables s, r, t_1, and t_2 with the pivot constraint:

$$\begin{bmatrix} s & -r & t_1 \\ r & s & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix} \tag{4.70}$$

With the constraint we can solve t_1 and t_2:

$$\begin{aligned} t_1 &= p_1 - sp_1 + rp_2 \\ t_2 &= p_2 - rp_1 - sp_2 \end{aligned} \tag{4.71}$$

We again place our variables in the vectorized system of Equation 3.30:

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} \\ r\mathbf{a} + s\mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.72}$$

We open t_1 and t_2, rearrange, and decompose left-hand side to a matrix A and a vector of the unknowns s and r. We end up with a system in the desired form $A\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\epsilon}$:

$$\begin{bmatrix} \mathbf{a} - p_1\mathbf{1} & -\mathbf{b} + p_2\mathbf{1} \\ \mathbf{b} - p_2\mathbf{1} & \mathbf{a} - p_1\mathbf{1} \end{bmatrix} \begin{bmatrix} s \\ r \end{bmatrix} = \begin{bmatrix} \mathbf{c} - p_1\mathbf{1} \\ \mathbf{d} - p_2\mathbf{1} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.73}$$

For convenience, let us denote:

$$\begin{aligned} \mathbf{a}' &= \mathbf{a} - p_1\mathbf{1} \\ \mathbf{b}' &= \mathbf{b} - p_2\mathbf{1} \\ \mathbf{c}' &= \mathbf{c} - p_1\mathbf{1} \\ \mathbf{d}' &= \mathbf{d} - p_2\mathbf{1} \\ \gamma &= \mathbf{a}'^\top\mathbf{a}' + \mathbf{b}'^\top\mathbf{b}' \end{aligned} \tag{4.74}$$

Hence, Equation 4.73 becomes:

$$\begin{bmatrix} \mathbf{a}' & -\mathbf{b}' \\ \mathbf{b}' & \mathbf{a}' \end{bmatrix} \begin{bmatrix} s \\ r \end{bmatrix} = \begin{bmatrix} \mathbf{c}' \\ \mathbf{d}' \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.75}$$

We again derive the Gramian, its inverse, and $A^\top\boldsymbol{\alpha}$:

$$A^\top A = \begin{bmatrix} \mathbf{a}'^\top & -\mathbf{b}'^\top \\ \mathbf{b}'^\top & \mathbf{a}'^\top \end{bmatrix} \begin{bmatrix} \mathbf{a}' & -\mathbf{b}' \\ \mathbf{b}' & \mathbf{a}' \end{bmatrix} = \begin{bmatrix} \gamma & \mathbf{0} \\ \mathbf{0} & \gamma \end{bmatrix} = \gamma I \tag{4.76}$$

$$(A^\top A)^{-1} = \gamma^{-1} I \tag{4.77}$$

$$A^\top \alpha = \begin{bmatrix} \mathbf{a}'^\top & -\mathbf{b}'^\top \\ \mathbf{b}'^\top & \mathbf{a}'^\top \end{bmatrix} \begin{bmatrix} \mathbf{c}' \\ \mathbf{d}' \end{bmatrix} = \begin{bmatrix} \mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}' \\ -\mathbf{b}'^\top\mathbf{c}' + \mathbf{a}'^\top\mathbf{d}' \end{bmatrix} \tag{4.78}$$

From Equation 4.71, Equation 4.77, and Equation 4.78 we can directly derive the estimates:

$$\begin{aligned} \hat{s} &= \gamma^{-1}(\mathbf{a}'^\top\mathbf{c}' + \mathbf{b}'^\top\mathbf{d}') \\ \hat{r} &= \gamma^{-1}(-\mathbf{b}'^\top\mathbf{c}' + \mathbf{a}'^\top\mathbf{d}') \\ \hat{t}_1 &= p_1 - \hat{s}p_1 + \hat{r}p_2 \\ \hat{t}_2 &= p_2 - \hat{r}p_1 - \hat{s}p_2 \end{aligned} \tag{4.79}$$

The estimates are not defined if $\gamma = 0$. Next, we inspect this special case further.

4.6.1 Special cases

The estimates are undefined if $\gamma = 0$. Let us open $\gamma = 0$ to get a better understanding:

$$\gamma = \sum_{i=1}^{n}(a_i - p_1)^2 + \sum_{i=1}^{n}(b_i - p_2)^2 = 0 \tag{4.80}$$

Because $x^2 \geq 0$ for any $x \in \mathbb{R}$, we can see that $\gamma = 0$ if and only if all domain points a_i, b_i and the pivot are equal. A scaling or a rotation that could move them elsewhere does not exist, so they all yield equal loss.

In this case, again due to simplicity, we choose the scaling factor $\lambda = 1$ and $\theta = 0$ to be the best choice. Therefore also $\hat{s} = 1$ and $\hat{r} = 0$, thus causing $\hat{t}_1 = p_1 - 1p_1 + 0p_2$ and $\hat{t}_2 = p_2 - 0p_1 - 1p_2$ both become zero, yielding the identity transformation.

4.6.2 Algorithm

An algorithm that estimates an optimal scaling and rotation transformation around a pivot is given in Listing 4.6. The structure is identical to both scaling and rotation algorithms in Section 4.2.2 and Section 4.3.2.

Listing 4.6: *Given a list of domain points* X*, a list of range points* Y*, and a pivot point* p*, this function returns optimal pivoted scaling-rotation matrix parameters* s*,* r*,* t1*, and* t2*. The code is written in Python.*

```
def estimate_scaling_rotation(X, Y, p):

    N = min(len(X), len(Y))

    p1 = p[0]
    p2 = p[1]
    a2 = b2 = 0
    ac = ad = bc = bd = 0
    for i in range(1, N):
        a = X[i][0] - p1
        b = X[i][1] - p2
        c = Y[i][0] - p1
        d = Y[i][1] - p2
        a2 += a * a
        b2 += b * b
        ac += a * c
        ad += a * d
        bc += b * c
        bd += b * d
```

```
g = a2 + b2

if g < epsilon:
    return 1, 0, 0, 0

s = (ac + bd) / g
r = (ad - bc) / g
t1 = p1 - s * p1 + r * p2
t2 = p2 - r * p1 - s * p2

return s, r, t1, t2
```

A single call with n point pairs executes $16n + 13$ basic arithmetic floating-point operations. A single call uses constant amount of memory regardless of n. These are further inspected in Chapter 6.

4.7 Translation-scaling-rotation estimation

The least constrained of the transformation estimators derived in this thesis is able to estimate an optimal transformation that transforms a sequence of domain points as close to a sequence of range points as possible by translating, scaling, and rotating.

We begin with the vectorized system given in Equation 3.30 with the full set of free parameters. Now, $s = \lambda \cos\theta$ and $r = \lambda \sin\theta$.

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} \\ r\mathbf{a} + s\mathbf{b} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.81}$$

We add zero vectors to emphasize that each equation is a linear combination of s, r, t_1, and t_2:

$$\begin{bmatrix} s\mathbf{a} - r\mathbf{b} + t_1\mathbf{1} + t_2\mathbf{0} \\ \mathbf{b}s + \mathbf{a}r + t_1\mathbf{0} + t_2\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.82}$$

We decompose the left-hand side to a matrix and a column vector and arrive to the desired form $A\boldsymbol{\beta} = \boldsymbol{\alpha} + \boldsymbol{\epsilon}$:

$$\begin{bmatrix} \mathbf{a} & -\mathbf{b} & \mathbf{1} & \mathbf{0} \\ \mathbf{b} & \mathbf{a} & \mathbf{0} & \mathbf{1} \end{bmatrix} \begin{bmatrix} s \\ r \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} + \boldsymbol{\epsilon} \tag{4.83}$$

To apply the linear least squares solution of Equation 3.22, we need to derive the Gramian $A^\top A$, its inverse, and $A^\top \boldsymbol{\alpha}$. As with the translation-scaling in Section

4.4, the Gramian is large, this time in $\mathbb{R}^{4\times4}$, and requires us to define simplifying notation. Thus, let:

$$\begin{aligned} u &= \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} \\ v &= \mathbf{a}^\top \mathbf{1} \\ w &= \mathbf{b}^\top \mathbf{1} \end{aligned} \tag{4.84}$$

We also note in advance that $\mathbf{1}^\top \mathbf{1} = n$ and that $\mathbf{x}^\top \mathbf{y} = \mathbf{y}^\top \mathbf{x}$ for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Now we are ready to open the Gramian:

$$\begin{aligned} A^\top A &= \begin{bmatrix} \mathbf{a}^\top & \mathbf{b}^\top \\ -\mathbf{b}^\top & \mathbf{a}^\top \\ \mathbf{1}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{1}^\top \end{bmatrix} \begin{bmatrix} \mathbf{a} & -\mathbf{b} & \mathbf{1} & \mathbf{0} \\ \mathbf{b} & \mathbf{a} & \mathbf{0} & \mathbf{1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} & -\mathbf{a}^\top \mathbf{b} + \mathbf{b}^\top \mathbf{a} & \mathbf{a}^\top \mathbf{1} + \mathbf{b}^\top \mathbf{0} & \mathbf{a}^\top \mathbf{0} + \mathbf{b}^\top \mathbf{1} \\ -\mathbf{b}^\top \mathbf{a} + \mathbf{a}^\top \mathbf{b} & \mathbf{b}^\top \mathbf{b} + \mathbf{a}^\top \mathbf{a} & -\mathbf{b}^\top \mathbf{1} + \mathbf{a}^\top \mathbf{0} & -\mathbf{b}^\top \mathbf{0} + \mathbf{a}^\top \mathbf{1} \\ \mathbf{1}^\top \mathbf{a} + \mathbf{0}^\top \mathbf{b} & -\mathbf{1}^\top \mathbf{b} + \mathbf{0}^\top \mathbf{a} & \mathbf{1}^\top \mathbf{1} + \mathbf{0}^\top \mathbf{0} & \mathbf{1}^\top \mathbf{0} + \mathbf{0}^\top \mathbf{1} \\ \mathbf{0}^\top \mathbf{a} + \mathbf{1}^\top \mathbf{b} & -\mathbf{0}^\top \mathbf{b} + \mathbf{1}^\top \mathbf{a} & \mathbf{0}^\top \mathbf{1} + \mathbf{1}^\top \mathbf{0} & \mathbf{0}^\top \mathbf{0} + \mathbf{1}^\top \mathbf{1} \end{bmatrix} \\ &= \begin{bmatrix} u & 0 & v & w \\ 0 & u & -w & v \\ v & -w & n & 0 \\ w & v & 0 & n \end{bmatrix} \end{aligned} \tag{4.85}$$

Its inverse $(A^\top A)^{-1}$ can be found by hand by using the block matrix inversion [25]. The method is especially suitable in the cases where the upper left quarter of the matrix is diagonal, which applies to our Gramian:

$$A^\top A = \left[\begin{array}{cc:cc} u & 0 & v & w \\ 0 & u & -w & v \\ \hdashline v & -w & n & 0 \\ w & v & 0 & n \end{array}\right] = \left[\begin{array}{c:c} U & V \\ \hdashline W & N \end{array}\right] \tag{4.86}$$

where $U, V, W, N \in \mathbb{R}^{2\times2}$. To inverse this block matrix, the following inversion formula can be applied [25]:

$$\begin{bmatrix} U & V \\ W & N \end{bmatrix}^{-1} = \begin{bmatrix} U^{-1} + U^{-1}V(N - WU^{-1}V)^{-1}WU^{-1} & -U^{-1}V(N - WU^{-1}V)^{-1} \\ -(N - WU^{-1}V)^{-1}WU^{-1} & (N - WU^{-1}V)^{-1} \end{bmatrix} \tag{4.87}$$

However, we skip the details and provide only the result of the inversion:

$$
\begin{aligned}
(A^\top A)^{-1} &= \begin{bmatrix} U & V \\ W & N \end{bmatrix}^{-1} \\
&= \gamma^{-1} \begin{bmatrix} n & 0 & -v & -w \\ 0 & n & w & -v \\ -v & w & u & 0 \\ -w & -v & 0 & u \end{bmatrix} \\
&= \gamma^{-1} \begin{bmatrix} n & 0 & -\mathbf{a}^\top \mathbf{1} & -\mathbf{b}^\top \mathbf{1} \\ 0 & n & \mathbf{b}^\top \mathbf{1} & -\mathbf{a}^\top \mathbf{1} \\ -\mathbf{a}^\top \mathbf{1} & \mathbf{b}^\top \mathbf{1} & \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} & 0 \\ -\mathbf{b}^\top \mathbf{1} & -\mathbf{a}^\top \mathbf{1} & 0 & \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} \end{bmatrix}
\end{aligned}
\tag{4.88}
$$

where

$$
\gamma = un - v^2 - w^2 = n(\mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b}) - (\mathbf{a}^\top \mathbf{1})^2 - (\mathbf{b}^\top \mathbf{1})^2 \tag{4.89}
$$

We continue by opening $A^\top \alpha$:

$$
A^\top \alpha = \begin{bmatrix} \mathbf{a}^\top & \mathbf{b}^\top \\ -\mathbf{b}^\top & \mathbf{a}^\top \\ \mathbf{1}^\top & \mathbf{0}^\top \\ \mathbf{0}^\top & \mathbf{1}^\top \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d} \\ -\mathbf{b}^\top \mathbf{c} + \mathbf{a}^\top \mathbf{d} \\ \mathbf{c}^\top \mathbf{1} \\ \mathbf{d}^\top \mathbf{1} \end{bmatrix} \tag{4.90}
$$

Finally, we have arrived to:

$$
\hat{\boldsymbol{\beta}} = \begin{bmatrix} \hat{s} \\ \hat{r} \\ \hat{t}_1 \\ \hat{t}_2 \end{bmatrix} = \gamma^{-1} \begin{bmatrix} n & 0 & -\mathbf{a}^\top \mathbf{1} & -\mathbf{b}^\top \mathbf{1} \\ 0 & n & \mathbf{b}^\top \mathbf{1} & -\mathbf{a}^\top \mathbf{1} \\ -\mathbf{a}^\top \mathbf{1} & \mathbf{b}^\top \mathbf{1} & \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} & 0 \\ -\mathbf{b}^\top \mathbf{1} & -\mathbf{a}^\top \mathbf{1} & 0 & \mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b} \end{bmatrix} \begin{bmatrix} \mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d} \\ -\mathbf{b}^\top \mathbf{c} + \mathbf{a}^\top \mathbf{d} \\ \mathbf{c}^\top \mathbf{1} \\ \mathbf{d}^\top \mathbf{1} \end{bmatrix} \tag{4.91}
$$

from which we can take out the desired transformation parameter estimates:

$$
\begin{aligned}
\hat{s} &= \gamma^{-1} \Big(n(\mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d}) - \mathbf{a}^\top \mathbf{1}\ \mathbf{c}^\top \mathbf{1} - \mathbf{b}^\top \mathbf{1}\ \mathbf{d}^\top \mathbf{1} \Big) \\
\hat{r} &= \gamma^{-1} \Big(-n(\mathbf{b}^\top \mathbf{c} - \mathbf{a}^\top \mathbf{d}) + \mathbf{b}^\top \mathbf{1}\ \mathbf{c}^\top \mathbf{1} - \mathbf{a}^\top \mathbf{1}\ \mathbf{d}^\top \mathbf{1} \Big) \\
\hat{t}_1 &= \gamma^{-1} \Big(-\mathbf{a}^\top \mathbf{1}(\mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d}) - \mathbf{b}^\top \mathbf{1}(\mathbf{b}^\top \mathbf{c} - \mathbf{a}^\top \mathbf{d}) + \mathbf{c}^\top \mathbf{1}(\mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b}) \Big) \\
\hat{t}_2 &= \gamma^{-1} \Big(-\mathbf{b}^\top \mathbf{1}(\mathbf{a}^\top \mathbf{c} + \mathbf{b}^\top \mathbf{d}) + \mathbf{a}^\top \mathbf{1}(\mathbf{b}^\top \mathbf{c} - \mathbf{a}^\top \mathbf{d}) + \mathbf{d}^\top \mathbf{1}(\mathbf{a}^\top \mathbf{a} + \mathbf{b}^\top \mathbf{b}) \Big)
\end{aligned}
\tag{4.92}
$$

The estimates are not defined if $\gamma = 0$. Next, we inspect this special case further.

4.7.1 Special cases

Trivially, if $n = 0$ we choose the identity to be the best choice.

If $\gamma = 0$ the estimates are not defined. To inspect the case, let us open $\gamma = 0$:

$$\begin{aligned}\gamma &= n(\mathbf{a}^\top\mathbf{a} + \mathbf{b}^\top\mathbf{b}) - (\mathbf{a}^\top\mathbf{1})^2 - (\mathbf{b}^\top\mathbf{1})^2 \\ &= n\sum_{i=1}^{n} a_i^2 + n\sum_{i=1}^{n} b_i^2 - \Big(\sum_{i=1}^{n} a_i\Big)^2 - \Big(\sum_{i=1}^{n} b_i\Big)^2 = 0\end{aligned} \tag{4.93}$$

Theorem 4.4.1 states that this is equivalent to all the domain points being equal. As translation is allowed, the identity would not be optimal. Our simplest best option is to keep the initial scaling $\lambda = 1$ and rotation $\theta = 0$ and translate the domain to the mean of the range as described in Section 4.1.

4.7.2 Algorithm

An algorithm that estimates an optimal composite translation, uniform scaling, and rotation is given in Listing 4.7. The structure is identical to the translation-scaling algorithm presented in Section 4.4.2.

Listing 4.7: *Given a list of domain points* X *and a list of range points* Y*, this function returns optimal translation-scaling-rotation matrix parameters* s, r, t1, *and* t2*. The code is written in Python.*

```
def estimate_translation_scaling_rotation(X, Y):

    N = min(len(X), len(Y))

    a1 = b1 = c1 = d1 = 0
    a2 = b2 = ac = ad = bc = bd = 0
    for i in range(1, N):
        a = X[i][0]
        b = X[i][1]
        c = Y[i][0]
        d = Y[i][1]
        a1 += a
        b1 += b
        c1 += c
        d1 += d
        a2 += a * a
        b2 += b * b
        ac += a * c
        ad += a * d
```

```
        bc += b * c
        bd += b * d

    g = N * a2 + N * b2 - a1 * a1 - b1 * b1

    if g < epsilon:
        if N == 0:
            return 1, 0, 0, 0
        return 1, 0, (c1 - a1) / N, (d1 - b1) / N

    acbd = ac + bd
    adbc = ad - bc

    s = (N * acbd - a1 * c1 - b1 * d1) / g
    r = (N * adbc + b1 * c1 - a1 * d1) / g
    t1 = (-a1 * acbd + b1 * adbc + a2 * c1 + b2 * c1) / g
    t2 = (-b1 * acbd - a1 * adbc + a2 * d1 + b2 * d1) / g

    return s, r, t1, t2
```

A single call with n point pairs executes $16n + 29$ basic arithmetic floating-point operations. A single call uses constant amount of memory regardless of n. These are further inspected in Chapter 6.

This concludes the derivation of our 7 estimation algorithms. Next we will look into what we can do with them.

5 APPLICATIONS

From the results presented in this thesis, we have created and published two open-source software packages, *Nudged* and *Taaspace*. They both are available at *GitHub*, a popular repository for open source projects, and easily installable via *NPM*, a package manager for JavaScript projects. See [26] for Nudged and [27] for Taaspace.

Nudged, available for JavaScript programming language, implements the 7 algorithms we have presented here. Nudged provides an easy-to-install and software package and a documented programming interface, along with a unit test suite and a set of multi-touch application examples. We describe the way how we implemented the multi-touch in Nudged in Section 5.1. Nudged also has a stripped down implementation in Python which was developed for the needs of spatial correction of eye tracking data. We explain the application to spatial correction further in Section 5.2.

Taaspace, available for JavaScript, is a zoomable user interface toolkit for web browsers. It depends vigorously on Nudged both in *geometric layout* and input gesture recognition. Taaspace provides an easy-to-install package with documentation, unit tests, and example applications. We describe the application to geometric layout in Section 5.3.

5.1 Multi-touch

Although we now have the estimators for each of our 7 types, one big question remains: how should we apply them on real-world devices equipped with a touch screen? Here we discuss and analyze the issue and propose a method to connect the finger movements to the estimators so that applying the estimates to geometric objects makes sense.

5.1.1 Concepts

For the task, let us first define a set of concepts. These concepts help us to model the touch interaction and manage it algorithmically within an application.

1. *Pointer*: a stylus, finger, mouse, or other type of pointer that provides a two-dimensional location.

2. *Pointer API*: an application programming interface through which pointer location information is emitted for the application.

3. *Target shape*: a geometric two-dimensional shape to where we are applying the transformation. For example a rectangular image.

4. *Bound pointer*: a pointer whose movement should affect the target shape. For example, in the drag and drop gesture the mouse becomes a bound pointer of the dragged shape when a mouse button is pressed down. The mouse pointer reverts back to the unbound state when the button is released.

5. *Pointer sampling frequency*: the frequency at which the pointer location update is emitted for the application through the pointer API.

6. *Pointer event*: a single sample of the pointer location emitted through the pointer API.

7. *Pointer session*: the life span from the first to the last pointer event of that pointer. For a finger, the session starts when the finger touches the screen and ends when the finger leaves. For a mouse, the session starts when the mouse is connected and ends when disconnected.

8. *Pointer identifier*: a name of the pointer. The identifier is unique for each pointer session.

With these concepts, we are now able to first describe the devices that our algorithms can be applied on and then how to apply them.

5.1.2 Applicable devices and interfaces

We require that the devices and their pointer APIs provide data about the pointers as follows:

1. The device samples the locations of pointers at constant interval. If a location of a pointer changes, the API notifies about it.

2. The first and last pointer event of a pointer session are distinguishable from the events between them. With that, we can be certain on which event the session started or ended.

3. The device can decide whether two pointer events are from the same pointer or not by giving each pointer an pointer identifier and letting each pointer event to known the identifier of the pointer that produced the event.

4. The API emits the pointer events in temporal order. In other words, events are emitted in the same order as the locations are measured.

The *Touch Events* recommendation by W3C [28] fulfills these requirements and thus are implemented in all modern mobile web browsers [29]. Therefore, we can more generally assume that these requirements are fulfilled by most if not all web-capable devices with a touch screen.

We also assume that devices are able to record more than two concurrent pointer sessions. Even though the algorithms handle two pointers as well, there would not be much gain when compared to the existing methods. Fortunately, most current mobile devices with a touch screen allow from 5 to 10 simultaneous touches. Some larger touch screens, such as ones produced by a company named MultiTaction, can track practically unlimited number of touch points [30].

Final requirement for the devices is that the pointer sampling frequency as well as the screen refresh frequency is high enough for smooth interaction. Any frequency higher than 25 Hz would be suitable. Fortunately, most touch screens fill this requirement and some, like Apple iPad Pro, even go as high as 120 Hz [31].

5.1.3 Challenges

At first, it might seem that integrating the pointer API and the algorithms would be straightforward. Just record where the bound pointers first were and where they traveled. Pass these two point sequences to one of the estimators as the domain and the range, receive the fitted transformation, and apply it to the target shape.

However, a first challenge in that approach is that the user would not receive direct feedback until the fingers become lifted. Therefore to provide a real-time experience, we should recompute and apply the transformation frequently enough if not each time when we receive a pointer event for one of the bound pointers. As a result, if the computational delay is low, the user can experience a smooth and reactive geometric transformation, alike to its physical counterpart.

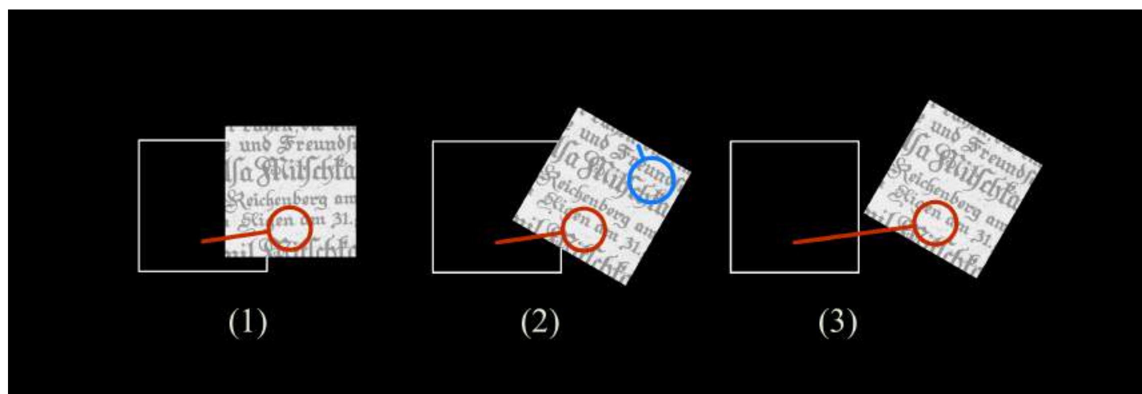

Figure 5.1: *Dynamics of the number of concurrent touch points. (1) At first, a user is translating a geometric object with one finger. (2) Then, during the translation, the user applies a small rotation to the object with a second finger. (3) The user lifts the second finger and should be able to continue the translation with the first.*

A second challenge is that in practice, the number of concurrent pointer sessions does not stay constant. New pointers might appear and some become removed during the transformation gesture. For example, a user might first translate an object by a single finger and then bring in a second finger to apply a small rotation along the way, as illustrated in Figure 5.1. The user most probably expects the rotation to happen because that is how objects would behave in the physical world. Therefore we must be able to react to these appearing and disappearing pointers.

5.1.4 Solution

Probably the simplest approach is to compare the previously know locations of bound pointers to their new locations. Each time the API notifies us that one or several pointers moved, we estimate a *microtransformation*. For the domain we use the sequence of previously known locations. For the range we update the sequence with the new locations. After the estimation, the sequence of previously locations is updated to match the range. Typically in practice the API emits pointer updates so frequently that only 1 or 2 pointers moved by only a few pixels, hence the prefix *micro*.

When a new pointer appears we add its location to a *dictionary* (also called an *associative array* or a *map*) of previously known locations by its pointer identity. Likewise, when a pointer disappears, it is removed from the dictionary. The addition is needed to make the domain and range include all the bound pointers. The removal is needed to remove the newly unbound pointers from the estimation. Otherwise they would appear as stationary, still ongoing pointers and thus the movements of the real bound pointers would have less effect than expected.

Listing 5.1: *This function enables multi-touch translation and rotation on a given geometric shape by using the microtransformation approach. The code is written in Python-like pseudocode and is based on how touch events are handled in web applications.*

```
def make_translatable_rotatable(shape):
    locations = {}

    def on_pointer_start(event):
        locations[event.id] = [event.x, event.y]

    def on_pointer_move(event):
        new_locations = locations.copy()
        new_locations[event.id] = [event.x, event.y]
        dom = locations.values()
        ran = new_locations.values()
        tr = estimate_translation_rotation(dom, ran)
        shape.transform_by(tr)
        locations = new_locations
```

```
    def on_pointer_end(event):
        if event.id in locations:
            del locations[event.id]

    shape.on('pointerstart', on_pointer_start)
    shape.on('pointermove', on_pointer_move)
    shape.on('pointerend', on_pointer_end)
```

The microtransformation approach can be implemented relatively simply as shown in Listing 5.1. We assume here that a geometric shape emits 3 types of pointer events: *pointerstart*, *pointermove*, and *pointerend*. The types are handled separately by 3 event handler functions. The previously known locations of each pointer are stored to a dictionary *locations* and they are fed into a transformation estimator as described above. The estimated small transformation is then applied to the shape. We assume that the update is then immediately presented on the screen. The estimator `estimate_translation_rotation` could be replaced by any of the 7 estimators we have presented, given that a pivot point is included if needed.

As a result, users are able to translate and rotate the shape and perceive it to follow the pointers in real-time in a way that would be expected with a lightweight physical shape on a frictionless surface. The approach does not place any restrictions on the number of concurrent pointers, the number of users, or even the number of shapes. Multiple users can simultaneously manipulate multiple shapes with multiple fingers, of course within the limits of the device and the pointer API.

During the development Nudged and Taaspace, we experienced with a few alternative approaches. However, we found the microtransformation approach to be the most practical due to its simplicity. The other approaches tried to overcome a minor drawback of the microtransformations, namely accumulation of residuals and floating-point rounding errors. For a general reference on the effects on rounding errors, see [32]. Because of the accumulation, the shape can slightly drift under the fingers during a uncommonly long-lasting and complex transformation. For example, if the shape is vigorously transformed for multiple seconds without lifting any fingers and then the fingers are moved back to their initial positions, the shape has slightly drifted from its initial position. However, the drift is so imperceptible that additional computational or programming demand of the other approaches were not justified.

5.2 Spatial correction

In addition to multi-touch, we have deployed the Python version of Nudged for eye tracker calibration at Infant Cognition Laboratory at University of Tampere (ICL) [33]. At ICL, eye trackers record the location of gaze on a screen. Often it happens that the participant is not in an optimal alignment with the camera or alternatively the screen and the camera have slightly moved. A calibration procedure is provided by the tracker manufacturer and usually run before actual measurements.

However, in the case of infants, the calibration procedure is often unsuccessful and the data needs to be gathered using a default calibration. The default calibration does provide precise but, with high probability, not accurate results. Fortunately, during the experiment there is moments when we known with a high certainty the exact position on the screen where to the child's focus is lured. By feeding these positions and corresponding measured positions into Nudged's translation-scaling-rotation estimator, we are able to sufficiently correct the other measured positions with the resulting transformation. One would still argue that a more general transformation estimation, including projective transformations, would be more suitable to eye tracking calibration. However, their performance would not necessarily be better due to increased number of free parameters, which could lead the model to overfit the data.

5.3 Geometric layout

The usefulness of Nudged algorithms to geometric layout was unexpected. With them, to move a shape to a place on two-dimensional space, we no longer needed to know the exact transformation that would do that. Before Nudged, with an earlier version of Taaspace, a lot of development time went to figuring out distances and angles between shapes and target places. This was tedious work and prone to error.

Figure 5.2: *In Taaspace, HTML elements can be translated, scaled, and rotated into nontrivial arrangements thanks to the algorithms presented in this thesis.*

Instead, now we only need to provide a sequence of points on the shape and a corresponding sequence of points on the target place. Given these two point sequences, Nudged is able to compute the required transformation. In other words, we only need to give an example of the results of the transformation without knowing the transformation itself. This *example-driven programming*, as we call it, has at least

initially shown to be an efficient way to implement even complex arrangements of content. See Figure 5.2 for an example where three varying sized rectangular HTML elements have been positioned to match their corners. Without Nudged, calculating the required transformations for such an arrangement would be anything but a trivial programming task.

6 ANALYSIS AND DISCUSSION

6.1 Computational efficiency

To avoid unwanted perception of delay, input devices such as mouses, styluses, and touch screens sample their location with a fast enough *sampling rate* [34]. Typical sampling rate (also *scan rate*) of a touch device is about 60 Hz to 120 Hz [35]. Therefore in practice, transformation estimation can be needed to be executed over 100 times a second. A computationally intensive algorithms would thus be unacceptable.

To analyze the efficiency we exploit two concepts commonly used to analyze algorithms. These are the *time complexity* and the *space complexity* where the former characterizes the computation time and latter the use of memory in relation to input size. They can reveal if an algorithm would turn out to be unusable in practice and especially with large input.

In addition, we should be aware if the algorithms internally call relatively time-consuming functions such as trigonometric functions. However, we already have seen that the algorithms consists of relatively simple, constant time operations, the square root being the most expensive of them. During iteration, only addition, subtraction, and multiplication are used, thus minimizing the effect of the division and the square root, which are considered more expensive although still very basic. We can safely conclude that the operations should not be a problem. [36]

6.1.1 Time and space complexity

For time and space complexity, we use the *big-O notation*. An algorithm that executes $3n^2 + n + 2$ operations to yield a result has a *quadratic time complexity* $O(n^2)$ where n is the input size. An algorithm that executes $2^{2n} + n^2$ operations has *exponential time complexity* $2^{O(n)}$. As we can perceive, only the term with the fastest growth matters. When n is large, as often is the case in practice, the term with the fastest growth will determine the computation time of the algorithm in the end. An algorithm that executes 2^n constant time operations in a single call would execute 1024 operations when $n = 10$ but when $n = 100$, the required about 10^{30} operations would make the algorithm totally unusable. Thus, the complexity matters. A good formal definition and introduction to computational complexity is given for example in [37].

Table 6.1: *Required number of arithmetic floating-point operations (FLOs), FLOs in a second (FLOPS), and time and space complexities for the algorithms.*

Algorithm	FLOs	Time	Space	FLOs (n=10)	FLOPS (n=10, f=120 Hz)
T	$4n + 4$	$O(n)$	$O(1)$	44	5280
S	$8n + 7$	$O(n)$	$O(1)$	87	10440
R	$12n + 16$	$O(n)$	$O(1)$	136	16320
TS	$12n + 34$	$O(n)$	$O(1)$	154	18480
TR	$12n + 27$	$O(n)$	$O(1)$	147	17640
SR	$16n + 13$	$O(n)$	$O(1)$	173	20760
TSR	$16n + 29$	$O(n)$	$O(1)$	189	22680

The space complexity of an algorithm, also described thoroughly in [37], gives the order of magnitude of additional memory units required to run the algorithm. For example, the arithmetic mean of a number sequence requires two memory units, one for the sum and another for the length of the sequence. Thus the space complexity of such algorithm is $O(1)$.

The number of arithmetic operations of our 7 algorithms and their time and space complexities are given in Table 6.1. The table also gives the number of operations required in a single call and during 1 second in 10-fingered multi-touch when estimation frequency of 120 Hz is assumed.

Table 6.1 shows us that all the algorithms have a *linear time complexity* [37, p. 253] and a constant space complexity. This is a very desirable result and ensures that the algorithms can be applied even on large data sets. In high-performance multi-touch interaction with 10 fingers, the heaviest of them requires 22680 floating-point operations per second (FLOPS). Given that the mobile processors in 2013 were capable of computing 10^9 FLOPS [38], this should consume only a fraction of the available computing power.

If the algorithms are written in an interpreted language, lots of computation is required in addition to the arithmetic operations. Even though the time and space complexities remain the same, the FLOPS in Table 6.1 might not tell us much about the real computational requirements. Therefore in Section 6.1.2, we will measure the performance of the algorithms when written in interpreted JavaScript and running on real-world hardware.

6.1.2 Running time on web browsers

To give an example of the efficiency of the algorithms, we conduct a benchmark on two web browsers. The results depend highly on the given environment, which includes features of hardware, operating system, programming language and other ongoing computational activity. Therefore we expect the results not to be generally conclusive but however give a representative order of magnitude of the running times.

The environment for the benchmark is the following. For hardware we have an Apple MacBook Air Mid 2012 laptop with an 1.8 GHz Intel Core i5 processor. The

laptop is connected to a power adapter. The benchmark will be run on two web browsers, Google Chrome version 49.0.2623.112 and Apple Safari 9.1 (11601.5.17.1). The browsers in turn run on Apple OS X 10.11.4. There is a few computationally insignificant applications running on the background.

For the benchmark, we measure the algorithms implemented in Nudged (See Chapter 5) which are written in JavaScript. We measure the running time of each algorithm separately for 5 varisized data sets. Each data set contain two point sequences, one for the domain and one for the range. The sizes are 10, 100, 1000, 10000, and 1000000 points per a point sequence. The points are generated randomly and uniformly into a unit square. For a benchmarking tool, we use *Benchmark.js* v2.1.0 [39]. At each execution cycle, Benchmark.js calls Nudged to estimate a given type of transformation between point sequences. The execution time is recorded and the execution repeated from 20 to 60 times depending on the variance.

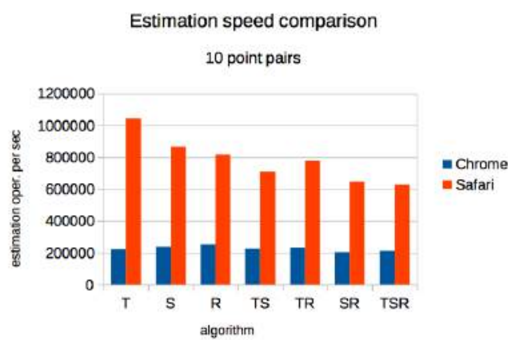

Figure 6.1: *Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 10 point pairs. A higher bar is better.*

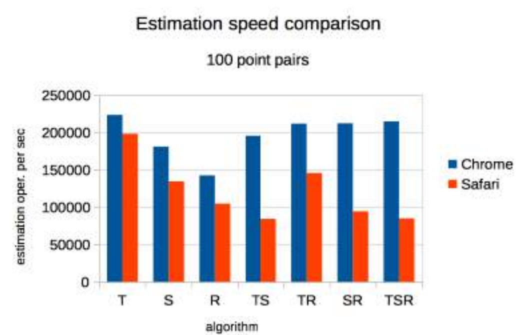

Figure 6.2: *Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 100 point pairs. A higher bar is better.*

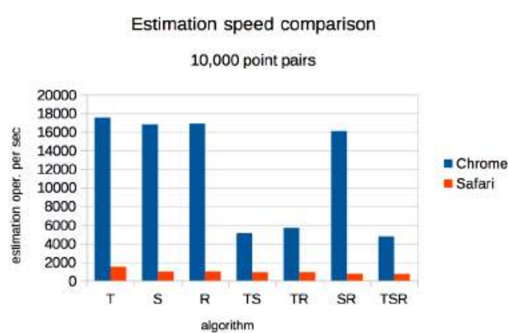

Figure 6.3: *Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 10,000 point pairs. A higher bar is better.*

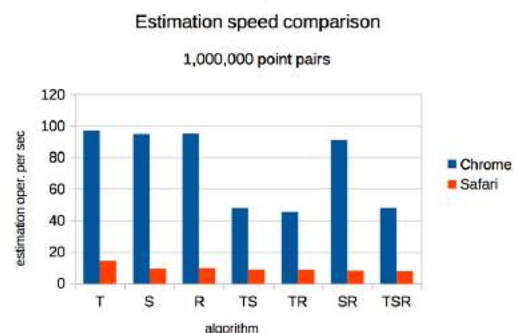

Figure 6.4: *Estimation computation speed differences between Google Chrome and Apple Safari and the 7 algorithms with 1,000,000 point pairs. A higher bar is better.*

We present the benchmark results here without going into much details. For multi-touch, most important results are given in Figure 6.1. With 10 points in the domain,

even the slowest result was over 200,000 estimations per seconds, which is 3 orders of magnitude higher than the high-end sampling rate of 120 Hz. On Chrome, interaction with 10,000 fingers would still be manageable from Nudged's perspective, as shown in Figure 6.3. All in all, it seems unlikely that the estimation would become a computational bottleneck even on low-end mobile devices.

The average execution times between the algorithms vary. As illustrated in Figure 6.1, Figure 6.2, Figure 6.3, and Figure 6.4 and as expected from the number of arithmetic operations, translation-scaling-rotation estimation is typically slower than plain translation estimation. Unexpectedly on Chrome, scale-rotation seems to be occasionally significantly faster than translation-scaling and translation-rotation whereas on Safari, translation-rotation seems the most efficient of the three. An interesting finding is also the difference between the web browsers. Chrome seems to do poorly with small sequences but outperform Safari by an order of magnitude when the sizes increase.

6.2 Robustness

The transformation gestures in example applications of Nudged and Taaspace have shown to work well and received positive comments from users. The users claimed the interaction to feel exceptionally natural and free from errors. Of course, a number of unbiased user experience studies are required to proof these claims. However, the examples already work as the proof-of-concept: the algorithms can be harnessed to provide a distinctive multi-touch experience.

Even though we have claimed the algorithms being robust in the sense of error-free interaction, in the sense of pattern recognition they are not. The least-squares method is known [14] to be vulnerable to outliers and this is also the case with our algorithms. A single unrealistic measurement can yield the result unusable. Fortunately, in our experience the touch APIs and devices provide the touch points in accurate manner. In eye tracking data calibration however, outliers are common.

A method to improve the robustness to outliers is called the iteratively weighted least-squares. Its application to transformation estimation is discussed for example by Haralick et al. [14] and Holland and Welsch [40]. In weighted least-squares, each sample point has a weight between 0 and 1 what determines its relative impact on the estimate. In iteratively weighted least-squares, the sample points are first equally weighted and estimation is conducted as in plain least-squares. The samples are then reweighted by their residuals so that the more they deviate from the estimate the smaller their weight. The estimation and reweighting phases are iterated until convergence which typically is rapidly achieved. As a consequence, the weights of the outliers tend to approach zero.

The multi-touch algorithm we presented in Section 5.1.4 have worked as expected in most of the situations. However, when the fingers are drawn together, we occasionally experience vibrations in the output geometry. We hypothesize this to be caused by the inability of the devices to recognize the closely packed fingertips as

separate fingers and repeatedly misjudging two tips as one. The caused unnatural interaction could be alleviated by detecting unrealistically quick finger movements especially when the fingers are close to each other.

A greater, and more general challenge for robust interaction is the variety of overlapping operating system level gestures that prohibit applications to benefit from more than 2 or 3 fingers. We discussed this issue in Section 2.3. During the development of Nudged and Taaspace, we repeatedly faced this challenge when testing the software on different devices. Nevertheless, even with 3 fingers the benefits are noticeable, not only because the equal respect but the robust handling of dynamics in the number of fingers.

7 CONCLUSION

In this thesis we have mathematically derived and compactly and software-developer-friendly represented 7 computationally robust and fast algorithms to estimate optimal nonreflective similarity transformations especially for purposes of geometric multi-touch manipulation. The algorithms let unlimited number of fingers simultaneously and equally being taken into account when geometric objects are being manipulated and by that remove limitations and pitfalls of traditional multi-touch transformation methods that are based on two fingers.

The algorithms are being implemented in a production-ready software library *Nudged* and already applied in a zooming user interface library *Taaspace* and at Infant Cognition Laboratory at University of Tampere. They have shown to be useful not only in multi-touch, but surprisingly in geometric layout and spatial data calibration.

Regardless of being already in use and computationally sound, the enhanced touch interaction they provide is not yet validated by user experience studies. Initial feedback shows promising but further studies on ergonomics and user error rates similar to [41] and [42] are needed to yield valid conclusions. Nevertheless, the hypothesis is that the multi-touch interaction methods made possible by the algorithms have the potential to supersede current publicly available methods. This is not only because the presented methods are sound and promising but because they are now made freely available for the public in well-documented and implementation-ready manner.

REFERENCES

[1] F. Ion. (2013-04). From touch displays to the surface: A brief history of touchscreen technology, [Online]. Available: http://arstechnica.com/gadgets/2013/04/from-touch-displays-to-the-surface-a-brief-history-of-touchscreen-technology/ (visited on 2016-05-11).

[2] B. Shneiderman, "Touch screens now offer compelling uses", *IEEE Software*, vol. 8, no. 2, pp. 93–94, 1991-03, ISSN: 0740-7459. DOI: 10.1109/52.73754.

[3] J. Nielsen. (1995). 10 usability heuristics for user interface design, [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/ (visited on 2016-04-10).

[4] MathWorks. (2016). Fit geometric transformation to control point pairs, [Online]. Available: http://se.mathworks.com/help/images/ref/fitgeotrans.html (visited on 2016-05-11).

[5] L. G. Brown, "A survey of image registration techniques", *ACM Comput. Surv.*, vol. 24, no. 4, pp. 325–376, 1992-12, ISSN: 0360-0300. DOI: 10.1145/146370.146374.

[6] J. M. Fitzpatrick, J. B. West, and C. R. Maurer, "Predicting error in rigid-body point-based registration", *IEEE Transactions on Medical Imaging*, vol. 17, no. 5, pp. 694–702, 1998-10, ISSN: 0278-0062. DOI: 10.1109/42.736021.

[7] Hammer.js. (2016). Hammer.js, [Online]. Available: http://hammerjs.github.io/ (visited on 2016-05-11).

[8] GitHub. (2015). Hammer.js issue #889 - feat(rotation-consistency): Don't rotate 180 degrees when initial finger is removed first, [Online]. Available: https://github.com/hammerjs/hammer.js/issues/889 (visited on 2016-05-11).

[9] W. Westerman, "Hand tracking, finger identification, and chordic manipulation on a multi-touch surface", PhD thesis, University of Delaware, 1999.

[10] Microsoft. (2016). Touch design guidelines, [Online]. Available: https://msdn.microsoft.com/en-us/windows/uwp/input-and-devices/guidelines-for-user-interaction (visited on 2016-05-11).

[11] Apple. (2016-03). iOS human interface guidelines: Design principles, [Online]. Available: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Principles.html (visited on 2016-05-11).

[12] Ask Ubuntu. (2016). Help to disable the multi-touch gestures in 14.04?, [Online]. Available: http://askubuntu.com/q/461481 (visited on 2016-05-08).

[13] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors", *Acta Crystallographica Section A*, vol. 34, no. 5, pp. 827–828, 1978-09. DOI: 10.1107/S0567739478001680.

[14] R. M. Haralick, H. Joo, C.-N. Lee, X. Zhuang, V. G. Vaidya, and M. B. Kim, "Pose estimation from corresponding point data", *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 6, pp. 1426–1446, 1989-11, ISSN: 0018-9472. DOI: 10.1109/21.44063.

[15] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration", *Medical Image Analysis*, vol. 2, no. 1, pp. 1–36, 1998, ISSN: 1361-8415. DOI: 10.1016/S1361-8415(01)80026-8.

[16] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 4, pp. 376–380, 1991-04, ISSN: 0162-8828. DOI: 10.1109/34.88573.

[17] J. H. Challis, "A procedure for determining rigid body transformation parameters", *Journal of Biomechanics*, vol. 28, no. 6, pp. 733–737, 1995, ISSN: 0021-9290. DOI: 10.1016/0021-9290(94)00116-L.

[18] D. C. Alexander, C. Pierpaoli, P. J. Basser, and J. C. Gee, "Spatial transformations of diffusion tensor magnetic resonance images.", *IEEE Trans. Med. Imaging*, vol. 20, no. 11, pp. 1131–1139, 2001. DOI: 10.1109/42.963816.

[19] L. E. Spence, A. J. Insel, and S. H. Friedberg, *Elementary Linear Algebra: A Matrix Approach*. Prentice Hall, 2000, ISBN: 0-13-716722-9.

[20] A. F. Möbius, *Der barycentrische Calcul*. Barth, 1827.

[21] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009, ISBN: 0136042597, 9780136042594.

[22] E. W. Weisstein. (2016). Least squares fitting–polynomial, MathWorld–A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html (visited on 2016-04-14).

[23] ——, (2016). Moore-penrose matrix inverse, MathWorld–A Wolfram Web Resource, [Online]. Available: http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html (visited on 2016-04-14).

[24] W. Alpha. (2016). Inverse of {{u,v,w},{v,n,0},{w,0,n}}, Wolfram Alpha LLC, [Online]. Available: https://www.wolframalpha.com/input/?i=inverse+of+%7B%7Bu,v,w%7D,%7Bv,n,0%7D,%7Bw,0,n%7D%7D (visited on 2016-05-13).

[25] T.-T. Lu and S.-H. Shiou, "Inverses of 2 x 2 block matrices", *Computers & Mathematics with Applications*, vol. 43, no. 1–2, pp. 119–129, 2002, ISSN: 0898-1221. DOI: 10.1016/S0898-1221(01)00278-4.

[26] GitHub. (2016). Nudged, [Online]. Available: https://github.com/axelpale/nudged (visited on 2016-05-08).

[27] ——, (2016). Taaspace, [Online]. Available: https://github.com/taataa/taaspace/ (visited on 2016-05-08).

[28] The World Wide Web Consortium (W3C). (2013). Touch events recommendation, [Online]. Available: `https://www.w3.org/TR/touch-events/` (visited on 2016-03-17).

[29] Mozilla Developer Network. (2016). Touch events, [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/API/Touch_events` (visited on 2016-03-17).

[30] MultiTaction. (2016). Technology, [Online]. Available: `https://www.multitaction.com/technology` (visited on 2016-03-17).

[31] Apple. (2016). iPad Pro - Apple Pencil, [Online]. Available: `http://www.apple.com/apple-pencil/` (visited on 2016-05-09).

[32] D. Goldberg, "What every computer scientist should know about floating-point arithmetic", *ACM Comput. Surv.*, vol. 23, no. 1, pp. 5–48, 1991-03, ISSN: 0360-0300. DOI: `10.1145/103162.103163`.

[33] University of Tampere. (2016). Infant cognition laboratory, [Online]. Available: `http://www.uta.fi/med/icl/index.html` (visited on 2016-05-09).

[34] R. S. Allison, L. R. Harris, M. Jenkin, U. Jasiobedzka, and J. E. Zacher, "Tolerance of temporal delay in virtual environments", in *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, ser. VR '01, Washington, DC, USA: IEEE Computer Society, 2001, pp. 247–, ISBN: 0-7695-0948-7.

[35] S. Hotelling, W. Chen, C. Krah, J. Elias, W. Yao, J. Zhong, A. Hodge, B. Land, and W. den Boer, *Integrated display and touch screen*, US Patent 9,268,429, 2016-02-23. [Online]. Available: `https://www.google.com/patents/US9268429`.

[36] V. Hindriksen. (2012-07). How expensive is an operation on a cpu?, StreamComputing BV, [Online]. Available: `https://streamcomputing.eu/blog/2012-07-16/how-expensive-is-an-operation-on-a-cpu/` (visited on 2016-05-14).

[37] M. Sipser, *Introduction to the Theory of Computation*, 2nd. Thomson Course Technology, 2006, ISBN: 0-534-95097-3.

[38] R. Garg. (2013). Exploring the floating point performance of modern ARM processors, [Online]. Available: `http://www.anandtech.com/show/6971/exploring-the-floating-point-performance-of-modern-arm-processors` (visited on 2016-05-09).

[39] M. Bynens and J.-D. Dalton. (2016). Benchmark.js v2.1.0, [Online]. Available: `https://benchmarkjs.com/` (visited on 2016-05-08).

[40] P. W. Holland and R. E. Welsch, "Robust regression using iteratively reweighted least-squares", *Communications in Statistics - Theory and Methods*, vol. 6, no. 9, pp. 813–827, 1977. DOI: `10.1080/03610927708827533`.

[41] E. Hoggan, M. Nacenta, P. O. Kristensson, J. Williamson, A. Oulasvirta, and A. Lehtiö, "Multi-touch pinch gestures: Performance and ergonomics", in *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ser. ITS '13, St. Andrews, Scotland, United Kingdom: ACM, 2013, pp. 219–222, ISBN: 978-1-4503-2271-3. DOI: `10.1145/2512349.2512817`.

[42] E. Hoggan, J. Williamson, A. Oulasvirta, M. Nacenta, P. O. Kristensson, and A. Lehtiö, "Multi-touch rotation gestures: Performance and ergonomics", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13, Paris, France: ACM, 2013, pp. 3047–3050, ISBN: 978-1-4503-1899-0. DOI: 10.1145/2470654.2481423.