DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

REASONING WITH SHAPES

by

Vahid JALILI

July, 2012

İZMİR

REASONING WITH SHAPES

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program

by
Vahid JALILI

July, 2012
İZMİR

M.Sc. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"REASONING WITH SHAPES"** completed by **VAHID JALILI** under supervision of **PROF. DR. SÜLEYMAN SEVİNÇ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Süleyman SEVİNÇ

Supervisor

[illegible]	[illegible]
Prof. Dr. Alp KUT	Prof. Dr. Haldun Karaca
(Jury Member)	(Jury Member)

[illegible]

Prof. Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

The research presented in this Master dissertation has been performed at the Dokuz Eylul University, Izmir, Turkey. Activities towards the completion of this thesis span two years of research and I wish to express my deepest appreciation and gratitude in my acknowledgments to all who have contributed and guided me through.

In the first place in my humble acknowledgment I would like to record my gratitude to Prof. Dr. Suleyman Sevinc which this thesis would not have been possible without his magnanimous supervision, advice and guidance from the very early stage of this research. My growth as a researcher was enriched and inspired by his passion and genius ideas in science and especially in Reasoning with Shapes. One could not wish for a better and friendlier supervisor.

I am as ever especially indebted to my beloved wife and parents for their love, pray, encouragements and indescribable support for my ambitions and aspirations. I simply cannot thank them enough. I also wish to thank my sister for her love and support during my studies.

It gives me great pleasure in acknowledging the support of Prof. Dr. Yalcin Cebi and Asst. Prof. Dr. Adil Alpkocak, also I wish to express my deep appreciation to all faculty members, dear colleagues and administrative staff of computer department for their support and valuable assistant in completion of my thesis.

Vahid JALILI

REASONING WITH SHAPES

ABSTRACT

Optimal logic determination between a set of shapes could be quite utile in computer vision. Investigation of Linear transformation in a set of shapes is a challenging topic and has wide range of applications, such as in Robotics, Aircraft and Satellite attitude determination and tracking systems. I propose a pictorial solution for linear transformation determination problem, in contrast to current optimal approaches that are benefiting from numerical roots.

I make abstractions of shapes and I try to determine the linear transformation between the set of shapes by using inexpensive Boolean logics. The nature of my solution decreases resource requirements and the complexity of a hardware implementation.

Keywords: Reasoning with shapes, computer vision, machine vision, shape abstraction, procrustes analysis.

ŞEKİLLERLE NEDENSELLEME

ÖZ

Şekiller arasındaki en uygun mantığın belirlenmesi bilgisayarla görmede oldukça yararlı olabilir. Şekiller kümesindeki doğrusal dönüşümün araştırılması ilgi çekici bir konudur ve robotik, uçak ve uyduların konumunu belirleme ve izleme sistemleri gibi geniş uygulama alanları vardır. Sayısal temellerden yararlanan şu anki optimum yaklaşımların aksine, doğrusal dönüşümün belirlenmesi problemine resimsel bir çözüm öneriyorum.

Ben şekilleri soyutluyorum ve az maliyetli Boolean lojiğini kullanarak şekiller arasındaki doğrusal dönüşümü belirlemeye çalışıyorum. Benim çözümümün doğası kaynak gereksinimini ve donanım uygulamasının karmaşıklığını azaltıyor.

Anahtar Kelimeler: Şekillerle nedenselleme, bilgisayarla görme, makine görmesi, şekil soyutlama, procrustes analizi.

CONTENTS

Page

M.SC THESIS EXAMINATION RESULT FORM ii

ACKNOWLEDGMENTS iii

ABSTRACT iv

ÖZ v

CHAPTER ONE – INTRODUCTION 1

1.1 Real World Experiences 1

1.2 Wahba’s Problem 5

1.2.1 Markley’s Methods 5

1.2.2 FOAM 6

1.2.3 SOMA 7

1.2.4 SVD – Based 8

1.3 Procrustes Analysis 9

1.3.1 Translation 10

1.3.2 Isotropic and Anisotropic Scaling 11

1.3.3 Rotation 11

1.3.4 Match Measurement 12

1.4 Kabsch Algorithm 13

CHAPTER TWO – NAÏVE APPROACH 15

2.1 Naïve Approach 16

2.2 Advantages and Disadvantages 20

2.3 Implementation 23

CHAPTER THREE – SEGMENTING – LEVELING APPROACH 26

3.1 Introduction 26

3.2 Segmentation.. 28

3.3 Segment Determination for a Landmark Point 32

3.3.1 Polar Coordinate System 32

3.3.2 Cartesian Coordinate System.. 33

3.4 Translation 34

3.5 Rotation... 35

3.6 Match Measurement 38

3.7 Leveling 42

3.8 Correctness Verification 46

CHAPTER FOUR - IMPLEMENTAION...50

CHAPTER FIVE – CONCLUSION AND FURTHER RESEARCH...........56

REFERENCES..60

APPENDIX 64

CHAPTER ONE
INTRODUCTION

1.1 Real World Experiences

Visual information that we gather from our environment plays an undeniable vital role with a vast range of applications in our life (even blind people visualize their environment in their own way), in continue I will mention some scenarios of this type that we all have experience of, but how we handle the situations is not completely clear to scientists yet.

Scenario A

A person is trying to cross a street; he\she looks at different sides of the street and if there is no possibility of accident regarding to speed and direction of cars movement, then crosses the street. This is a simple case of our interaction with environment; an algorithmic look of the scenario could be as following:

1. Determine your crossing path and minimum time (T_1) needed to pass
2. Determine the cars coming your direction
3. Find the closest car to you in the set of cars coming your direction
4. Estimate its speed and distance from you
5. Calculate: “how long it takes for the car to reach your path with its current speed? (T_2)”
6. If $T_1 < T_2$ or for more caution $T_1 \ll T_2$ then you can cross (if you have checked for an acceptable number of cars, if not goto (3) to check for more cars) else you cannot cross the street.

To fulfill each of the steps of this algorithm we need some knowledge of physics and some special devices to determine speed and distance. But can we argue that a child (grown enough) has the knowledge and the equipment that can cross the street safely? Surely not, it is analyzing the information (e.g. Visual information) which we gather from environment that helps us to conclude whether we can safely cross the street or not.

Scenario B

Does the person in Figure 1.1 can touch the wall or not?

If we know the values of X, X_1 and X_2 we could say the person can touch the wall if $X_2 = 0$ or if $X = X_1$. Laser-based or Radar/Sonar-based distance measurement equipment can accurately determine X, X_1 and X_2, but in case we don’t have access to these tools, we can estimate the values; but even for estimation we need to measure the distance somehow. However, in our life we can answer such questions with no need to these tools, just by using the visual information we receive from our eyes.

Suppose a person who is standing on top of a rock; with a level of confidence the person knows whether he\she will survive if jumps down. This issue is a challenging problem in robotics, although some notable approaches are available (Mondragón (2010)) but they are costly and most of them requires special equipment and aid such as GPS antenna and height information. Despite of us, even animals are capable of making this estimation without using such equipment that reveals the simplicity of this task and importance of visual information.

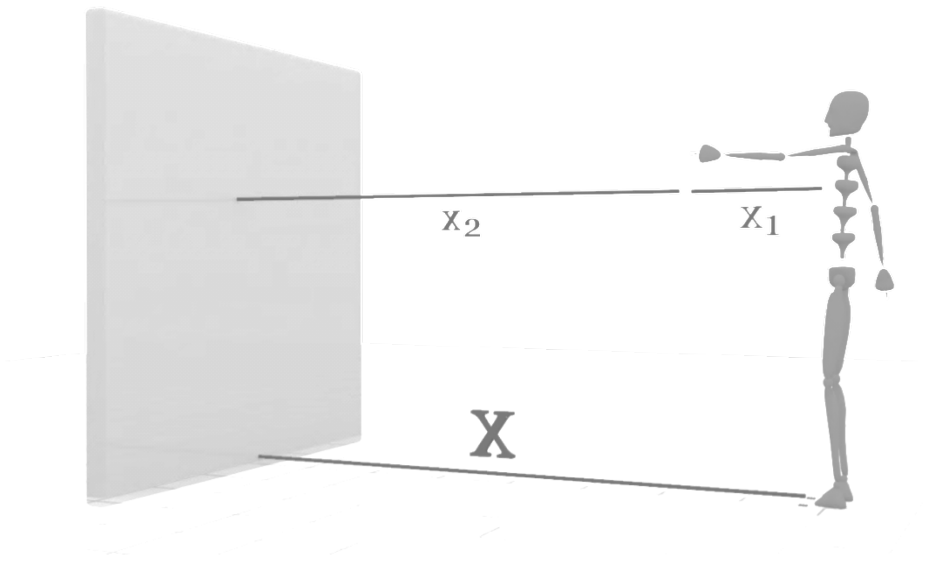


Figure 1.1 Distance estimation by using visual information

Scenario C

In crowd we can distinguish the people whom we know (e.g. our parents, wife and children) even if there are some other people who so look like each other, but if we are asked to draw their faces we may not be able to, unless we have some painting skills. But in computer science, if using an algorithm and a database we can distinguish a person; we can use the same database and another algorithm to draw the person’s face.

Scenario D

If we are given Figure 1.2 A and we are asked to group the points in the figure, we could easily do the task, but what if the same figure is given to a computerized program and the same question is asked? It might be a time consuming problem regarding to the number of points we have in the figure (note that the cardinality of points do not affect the complexity of the problem for us), one algorithm could be:

1. Determine the distance between all close pairs.
2. Calculate the mean distance.
3. Using mean value define a threshold for distance between pairs of same group.
4. Using the threshold determine which points are in a same group.

Although it is not clearly known yet that how we solve the problem, but with our experiences we guess the procedure demonstrated in Figure-2-B, might be our method to solve the problem.

Currently available approaches can solve these scenarios and many more scenarios of this type with accuracy, but for most cases we may prefer to sacrifice this accuracy to achieve tolerable approximate answers in less time with dependency to least and simplest equipment.

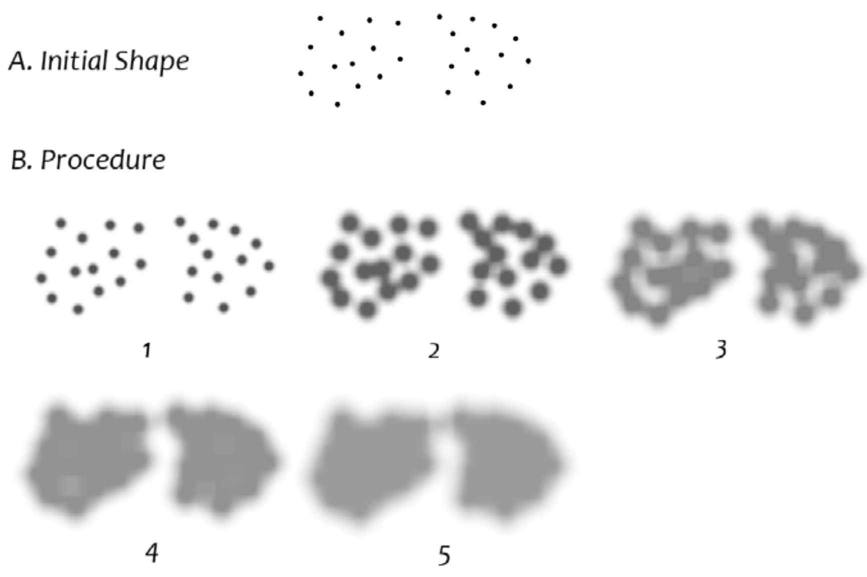


Figure 1.2 To group the points in initial shape, the demonstrated Procedure could be an option. The 5th shape in the Procedure shows two distinct shapes where the outline of these shapes is the border that environs the two groups of points.

In scenario A, the person may not notice the plate number of each car, or the clothes the drivers are wearing or even the exact model or color of cars; this does not mean that the person did not see any of these. For example, the person saw the plate number, but plate number was not important for the action he wanted to take at the time, hence he simply avoided it, but if he is asked to take the plate number of the cars in same situation instead of crossing the street, he may be able to read the plate number, and the same goes for the color and model of the cars and the clothes the driers are wearing.

This experience uncovers the role and necessity of abstractions we make for essential data out of full visual information we receive from our eyes. By abstracting we reduce the level of details to come up with less data to deal with, and I will use this advantage in my approach.

1.2 Wahba's Problem

Grace Wahba in 1965 defined a problem as follows:

Given two sets of n points $\{v_1 , \; v_2 , \dots , v_n\}$ *, and* $\{v_1^* , v_2^* , \dots , v_n^*\}$ *, where* $n \geq 2$ *, find the rotation matrix M (i.e., the orthogonal matrix with determinant +1) which brings the first set into the best least squares coincidence with the second. That is, find M matrix which minimizes* $\sum_{j=1}^{n} \left| v_j^* - M v_j \right|^2$ *(Wahba (1965))*

According to Wahba, solutions for the problem are mainly used in satellite attitude determination; in addition, some other applications in Robotics (e.g. Bruzzone & Callegari (2010)) and Tracking systems are defined. About a decade later Paul Davenport proposed an optimal solution for Wahba's problem, known as q-method, he himself did not publish his method, but the method is explained in details in (Markley & Mortari, M. (2000)) and (Shuster & OH (1981)), and according to (Fallon & Harrop & Sturch (1979)) NASA used this method to support HEAO missions. Another well-known solution is QUEST (QUaternion ESTimator), also some other solutions are proposed, such as (Shuster & Natanson (1993)), (Keat (1977)), (Shuster (1978)), (Mortari (1997)) and (Mortari (2000)).

1.2.1 Markley's Methods

F. Landis Markley in (Markley (1993)) presented FOAM (Fast Optimal Attitude Matrix) and SOMA (Slower Optimal Matrix Algorithm) and in (Markley (1988)) a Singular Value Decomposition (SVD) based solution for Wahba's problem.

Markley rewrites Wahba's non-negative loss function as follows:

$$L\,(\,A\,) = \frac{1}{2} \sum_{i=1}^{n} a_i \, |b_i - A r_i|^2 \qquad (1.2.1.1)$$

Where A is an orthogonal matrix to minimize the above loss function, n is the number of observations, a_i are positive weights and b_i and r_i are unit vectors representing corresponding observations in spacecraft body frame and unit vectors that are directions to some observed objects in reference frame, respectively.

Markley used some matrix manipulations and rewrite (1.2.1.1) as follows:

$$L\,(\,A\,) = \lambda_0 - Trace\,(\,A\,B^T\,) \qquad (1.2.1.2)$$

$$\lambda_0 = \sum_{i=1}^{n} a_i \qquad (1.2.1.3)$$

$$B = \sum_{i=1}^{n} a_i\, b_i\, r_i^T \qquad (1.2.1.4)$$

1.2.2 FOAM

Markley defines an iterative solution for λ and names it as FOAM, the procedure of his method is as following:

1. Normalize input observation and reference vectors
2. Calculate λ_0 (for normalized weights we have $\lambda_0 = 1$, Markley also solved λ_0 for different conditions) and B
3. Calculate the following scalars:

$$\det B\ , \qquad \|B\|^2\ , \quad \|Adjoint\ \ B\|^2 \qquad (1.2.2.1)$$

4. Compute λ as following:

$$\lambda_i = \lambda_{i-1} - \frac{\psi\ (\,\lambda_{i-1}\,)}{\psi'\,(\,\lambda_{i-1}\,)}\ , \qquad i = 1\,,2\,,\ldots \qquad (1.2.2.2)$$

$$\psi\,(\,\lambda\,) = (\,\lambda^2 - \|B\|^2\,)^2 - 8\lambda\ \det B - 4\|Adjoint\ \ B\|^2 \qquad (1.2.2.3)$$

$$(\,\psi'\ is\ derivative\ of\ \psi\,) \qquad \psi'\,(\,\lambda\,) = 4\,\lambda\,(\lambda^2 - \|B\|^2) - 8\det B \qquad (1.2.2.4)$$

5. Solve the following to obtain optimal attitude estimation:

$$Optimal\ A = \frac{(\zeta + \|B\|^2)\, B + \lambda\, Adjoint\ B^T - B\, B^T B}{\Gamma} \qquad (1.2.2.5)$$

$$\zeta = \frac{1}{2}\,(\lambda^2 - \|B\|^2) \qquad (1.2.2.6)$$

$$\Gamma = \zeta\,\lambda - \det B = \frac{\lambda}{2}\,(\lambda^2 - \|B\|^2) - \det B \qquad (1.2.2.7)$$

1.2.3 SOMA

Before I continue with Markley's SOMA method, I rewrite matrix B using SVD and diagonal values of Σ as follows:

$$\begin{gathered} B = \mathrm{M}\,\Sigma\,\mathrm{N}^T \\ S_1 \geq S_2 \geq |\,S_3\,| \end{gathered} \qquad (1.2.3.1)$$

Besides, Markley's SOMA method which is in the form of an analytical solution for S_1, has the following steps:

1. Normalize input observation and reference vectors *(Same as FOAM)*
2. Calculate λ_0 (for normalized weights we have $\lambda_0 = 1$, Markley also solved λ_0 for different conditions in (Markley (1993))) and B *(Same as FOAM)*
3. Calculate the following scalars: *(Same as FOAM)*

$$\det B\ , \qquad \|B\|^2\ , \quad \|Adjoint\ \ B\|^2 \qquad (1.2.3.2)$$

4. Compute λ as following:

$$\lambda = S_1 + (\,S_2 + S_3\,) \qquad (1.2.3.3)$$

$$S_2 + S_3 = \sqrt{\frac{\| \; Adjoint\; B \; \|^2 - \left(\frac{\det B}{S_1} \right)^2}{S_1^2} + \frac{2 \det B}{S_1}} \qquad (1.2.3.4)$$

$$S_1^2 = \frac{1}{3} \left(\|B\|^2 + 2\alpha \, cos \left(\frac{1}{3} \; cos^{-1} \left(\frac{\beta}{\alpha^3} \right) \right) \right) \qquad (1.2.3.5)$$

$$\alpha = \sqrt{\|B\|^4 - 3 \, \|Adjoint\; B\|^2} \qquad (1.2.3.6)$$

$$\beta = \|B\|^6 - \frac{9}{2} \left(\|B\|^2 \;\; \|Adjoint\; B\|^2\right) + \frac{27}{2} \left(\det B\right)^2 \qquad (1.2.3.7)$$

5. Optimal attitude matrix determination is same as FOAM.

1.2.4 SVD – Based

Markley proposes the following steps for his SVD – based solution for Wahba's problem.

1. From (1.2) calculate B
2. Calculate SVD of B. ($B = USV^T$)
3. Calculate d as following:

$$d = \det U \; \times \det V \qquad (1.2.4.1)$$

4. Compute optimal matrix A as follows:

$$A_{opt} = \; U \; [\; diag \; (1 \, , 1 \, , d \;) \;] \;\; V^T \qquad (1.2.4.2)$$

5. Computed minimized loss function as follows:

$$L \left(A_{opt}\right) = 1 - \; s_1 - \; s_2 - ds_3 \qquad (1.2.1.5)$$

Where $s_1 \geq \; s_2 \geq \; s_3$ are the diagonal values of S.

1.3 Procrustes Analysis

In Greek mythology, a character exist named Procrustes (or the stretcher, Prokoptas or Damastes); he was a bandit who used to suite his victims to his iron bed by racking, hammering or amputation. His fashion forms a root in statistical shape analysis, which is called **Procrustes Analysis**. In Procrustes analysis, two or more shapes are considered as identical if after **Procrustes Superimposition,** which is by applying transformations such as Translation, Uniform/Un-Uniform Scaling, Rotating and Reflection, the shapes coincide, if they do not coincide, or better say, if their **Procrustes Distance** is not zero, then their similarity is measured by the value of Procrustes Distance.

If all transformations are checked in Procrustes superimposition, then it is called **Full Procrustes Superimposition (FPS)**, and when scaling is not included, it is referred as **Partial Procrustes Superimposition (PPS)**.

Notable solutions for Procrustes Analysis are available, of those, some methods presented in (Gover & Dijksterhuis (2004)) and (Thomas (2006)). In reference (Gover & Dijksterhuis (2004)) an overall of some notable previous works is provided, here I summarized some parts of that information in a tabular format in Table 1.1; interested readers are advised to refer to reference (Gover & Dijksterhuis (2004)) for more details.

Table 1.1 Brief specification of some notable works regarding to Procrustes Analysis are given.O: Orthogonal Matrix, P: Orthogonal Projection Matrix, S: Least Square, G: Group Average, I: Inner Product

Comment	Author	Year	Sets count	O	P	S	G	I
Full Rank Matrices	Green	1952	2	*		*		
Deficient Rank case	Schonemann	1966	2	*		*		
Two-Sided	Cliff	1966	2		*			*
Pairwise	Gower	1971	K	*		*		
PINDIS	Borg, Lingoes	1978	K		*	*		
	Ten Berge, Knol	1984	K		*	*		*
	Peay	1988	K		*		*	
	Dijksterhuis, Gower	1991	K	*	*	*	*	

In statistical shape analysis, Procrustes analysis for two inputs is defined as to solve the following expression for T that minimizes the statement:

$$\|V_1T - V_2\| \tag{1.3.1}$$

Where V_1 is the traveler, T is the Procrustes superimposition, V_2 is the iron bed and $\|X\|$ means Euclidean/Frobenius norm $trace(\,A'A\,)$, the sum-of-squares of elements of A. In other words, V_1 is **shapes 1**, V_2 is **shape 2** and T is the transformation which if applied on V_1 results V_2 in case that the two shapes are identical.

1.3.1 Translation

As first step of Procrustes superimposition, we start with simplest transformation which is Translation. To be able to compare a set of shapes, all shapes must be transformed into an identical coordinate so that the centroid of all shapes coincides, this coordinate could be whether the center of coordinate system or any specific coordinate. We could write the translation applied form of (1.3.1) as following:

$$\|(V_1 - 1t_1')\,T - (V_2 - 1t_2')\| \tag{1.3.1.1}$$

We denote a matrix whose column vectors are one by 1. Since only proportional position of origin is important for us, from (1.3.1.1) we could have the following:

$$t' = t_1'\,T - t_2' \tag{1.3.1.2}$$

That by substituting (1.3.1.1) in (1.3.1.2) we will have:

$$\|V_1T - 1t' - V_2\| \tag{1.3.1.3}$$

And in case the equation (1.3.1) is preceded by a weighting matrix, that is:

$$||WV_1T - V_2|| \tag{1.3.1.4}$$

Where W is the weighting matrix; the translated form of (1.3.1.4) will be:

$$||\ W\ (V_1 -\ 1t_1'\ )\ T -\ (V_2 - 1t_2'\ )\ || \tag{1.3.1.5}$$

Interested readers in the generalized form of (1.3.1.5) are advised to refer to reference (Gover & Dijksterhuis (2004)) for more details.

1.3.2 Isotropic and Anisotropic Scaling

The second step of Procrustes superimposition is as simple as the first step; for the cases that T is not constrained by any means, an isotropic scaling factor is a scalar which we denote by S that is multiplied to V_1 as following:

$$||SV_1T - V_2|| \tag{1.3.2.1}$$

Note that, S is a scalar in isotropic scaling so it can be ingested into T , but for anisotropic scaling, S is a matrix that despite unlike isotropic scaling it can’t be assimilate into T , the order of multiplication is also important, that is:

$$||V_1ST -\ V_2||\ \neq\ ||V_1TS -\ V_2|| \tag{1.3.2.2}$$

1.3.3 Rotation

In case that T is not constrained by any conditions, we can solve (1.3.1) for T as follows:

$$T = \frac{V_1{}'\,V_2}{V_1{}'\,V_1} \tag{1.3.3.1}$$

We can write T in a SVD form, doing this, we can result:

$$T = U\Sigma W' \rightarrow \; \| V_1 U\Sigma W' - V_2 \| = \|V_1 U\Sigma - V_2 W\| \qquad (1.3.3.2)$$

This change in format means rotating V_2 to a position that matches rotated V_1 along with scaling. For conditions where any restriction(s) is (are) on T , please refer to reference (Gover & Dijksterhuis (2004)) - Chapter 13.

1.3.4 Match Measurement

Best match for (1.3.1) maybe measured using different criteria such as Correlation and Inner product, Least Squares Criteria and Matching of Rows and Columns .To measure match using Least Squares, in case T is not constrained in anyway, we can treat V_1 and V_2 symmetrically by doing either given (a) or (b):

a. **Two-Sided variant:** Solve (1.3.4.1) for V_1 and V_2 where both matrices have same number of columns. The resulting from (1.3.4.1) is trivial null solution.

$$S = \|V_1 T_1 - V_2 T_2\| \qquad (1.3.4.1)$$

b. We can rewrite (1.3.1) as (1.3.4.2), here the point is, (1.3.4.2) deem to be generalization of Procrustes problem when we have no apprehension of target matrix.

$$\frac{1}{4} S = \left\| V_1 T - \frac{1}{2}(V_1 T + V_2 T) \right\| = \left\| V_2 T - \frac{1}{2}(V_1 T + V_2 T) \right\| \qquad (1.3.4.2)$$

$$\frac{1}{2} S = \sum_{n=1}^{2} \left\| V_n T - \frac{1}{2}(V_1 T + V_2 T) \right\| \qquad (1.3.4.3)$$

1.4 Kabsch Algorithm

Wolfgang Kabsch propose an orthogonal solution for orthogonal partial Procrustes problem in (Kabsch (1976)) and (Kabsch (1978)), by an SVD based solution he tries to minimize the Root Mean Squared Deviation (RMSD) of two input sets, when determinant of rotation matrix is one. Unlike some solution for Procrustes problem that finds rotation around a single axis, Kabsch algorithm calculates transformation to a different orthonormal basis. Although Kabsch algorithm only calculates Rotation, but before checking for rotation, it requires a translation operation to coincide the centroid of the inputs.

Kabsch algorithm applies on two shapes where each shape is represented by a set of points. A sample set of points of a shape (Shape 1) is illustrated in Figure 1.3, the set consisting of two, three and in general, d columns presents the shape in a two dimensional $(2D)$, three dimensional $(3D)$ and d-dimensional coordinate system respectively. n Is the number of points presenting the shape, more points gives more details about the shape, which can increase the accuracy of the transformation determination and match measurement process.

Different methods for extraction of points from shapes are available in literature, of this landmarks are notable, triple classify of landmarks are: (I) Anatomical Landmarks, (II) Mathematical Landmarks and (III) Pseudo Landmarks

Shape 1

(P : Point)

2 Dimensional

P1	X_1	Y_1
P2	X_2	Y_2
P3	X_3	Y_3
	…	…
P n	X_n	Y_n

3 Dimensional

P1	X_1	Y_1	Z_1
P2	X_2	Y_2	Z_2
P3	X_3	Y_3	Z_3
	…	…	…
P n	X_n	Y_n	Z_n

d Dimensional

P1	D_{11}	D_{21}	…	D_{1d}
P2	D_{21}	D_{22}	…	D_{2d}
P3	D_{31}	D_{32}	…	D_{3d}
	…	…	…	…
P n	D_{n1}	D_{n2}	…	D_{nd}

Figure 1.3 A sample set of Points presenting a shape (Shape 1)

Kabsch algorithm runs in five steps as following (V : Shape 1 and W : Shape 2):

1. Translation
2. Calculate A from the equation:

$$A = V^T W \tag{1.4.1}$$

3. Calculate the SVD of A

$$A = M \, \Sigma \, N^T \tag{1.4.2}$$

4. Calculate optimal rotation matrix R

$$R = N \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \eta \end{pmatrix} M^T \tag{1.4.3}$$

$$\eta = Sign\,(\det A)$$

CHAPTER TWO
REASONING WITH SHAPES
NAÏVE APPROACH

Grace Wahba defined a problem and some novel approaches are proposed and are widely used; the Procrustes problem is almost the same as the problem that Wahba defined, only that in Procrustes analysis we have a more pictorial definition of the problem than Wahba's problem. Some novel algorithms such as Kabsch algorithm are available for Procrustes problem that are practical and in use.

An infant has intuitive understanding of cardinality and shape of objects and could understand whether two objects are the same but only transformed (Rotated, Scaled, symmetrized) or the objects are different (Izard & Dehaene-Lambertz & Dehaene (2008)). It would not be wise if we argue that a child has knowledge of the numerical solutions instinctively; because, not even infants but all people was able to manipulate objects even before mathematics exists; the tools and painting found inside caves that are dated back to thousands years ago is a proof.

My goal is to solve the problem in more pictorial root with least possible dependency on mathematics than Procrustes analysis roots. To achieve this goal, first I redefine the problem with some tunings as follows:

1. Given the set :

$$\{ Shape_1 \, , \; Shape_2 \, , \dots \; , \; Shape_n \}$$

a. Determine the transformation T between any pairs as:

$$T_m = \langle \; Shape_{\,m} , \; Shape_{\,m+1} \; | \, m \; \in \{ \, 1 \, , 2 \, , \dots \, , n-1 \, \} \rangle$$

Where $T \; \in \; (\, Rotation \, , Translation \, , Scale \,)^*$

b. Ensure the candidates to be $Shape_{\,n+1}$ by applying highly probable Ts' from $\{ \, T_1 \, , \; T_2 \; , \dots \, , \; T_{n\,-\,1} \}$ on $Shape_{\,n}$ in a descending order (i.e. The candidates with higher probabilities comes first and candidates with lower probabilities comes last)

2. *(Auxiliary)* Find $Shape_m$ in $Shape_n$

Of the ideas addressing the mentioned problems, a RAW approach seemed useful to me and I checked it in more details. Although the naïve approach that I checked was not acceptable at first because of the lack of any advantages over previous approaches despite of its reasonably fast and accurate results with some optimizations that I made during programing; but it worth reviewing because it is the base of the **Segmenting–Leveling** approach. In continue we will have a brief description of the naïve approach and next we will continue with **Segmenting–Leveling** approach in Chapter Three.

2.1 Naïve Approach

I use $x \times y$ matrices with entries $\{ 0 , 1 \}$ to present shapes in this naïve approach, as illustrated in Figure 2.1 for a sample shape, the values of black cells are 1 and white cells are 0.

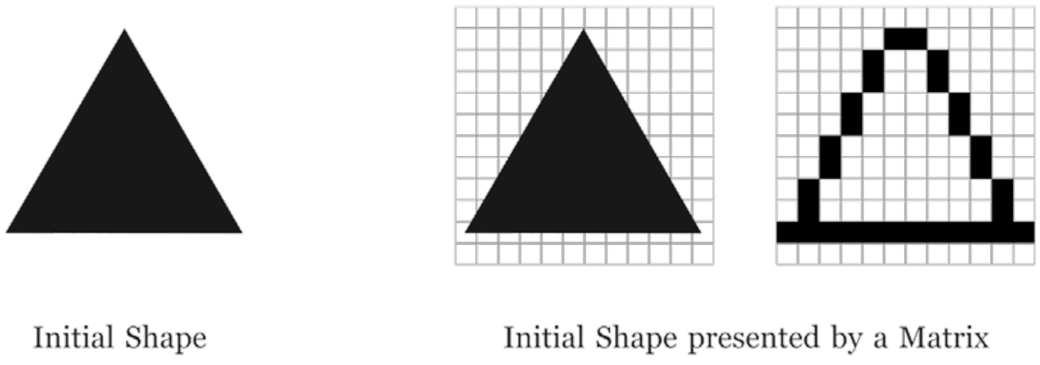
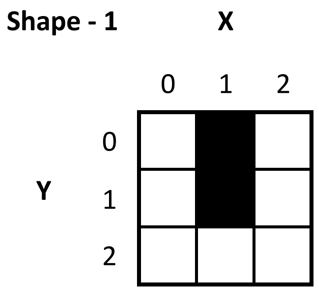


Figure 2.1 Presenting a sample shape using a matrix with values $\{ \mathbf{0} , \mathbf{1} \}$

As a naïve approach, in order to ascertain T for any pair I will do a state space search in a discreet coordinate system using a tree structure (illustrated in Figure 2.2). As an example, translation check for a **single rotation angl**e of two very simple shapes presented by 3×3 matrices is illustrated in Figure 2.3. The example in Figure 2.3 belongs to 0 degrees rotation, for the rest of the rotations, first I will rotated shape-1 then I will check for Translation as in Figure 2.3, meaning that I will repeat the process demonstrated in Figure 2.3 for 259 times and each time using θ degrees rotated of shape-1.

To optimize the process, only coordinates of incidences are stored and transformations are applied on the incidences, as following for the given sample shape (Shape – 1):



	X	Y
Point 1	1	0
Point 2	1	1

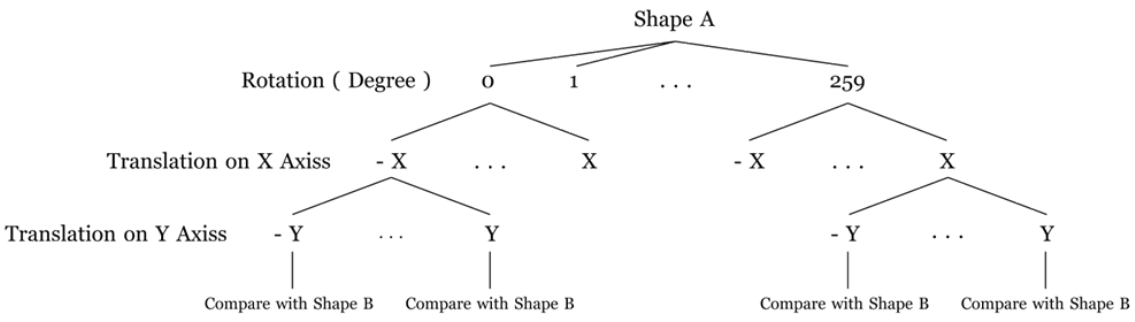


Figure 2.2 State space search for the transformation determination between Shape-A and Shape-B is given. “X” and “Y” are the number of columns and rows of the matrix presenting the shapes respectively.

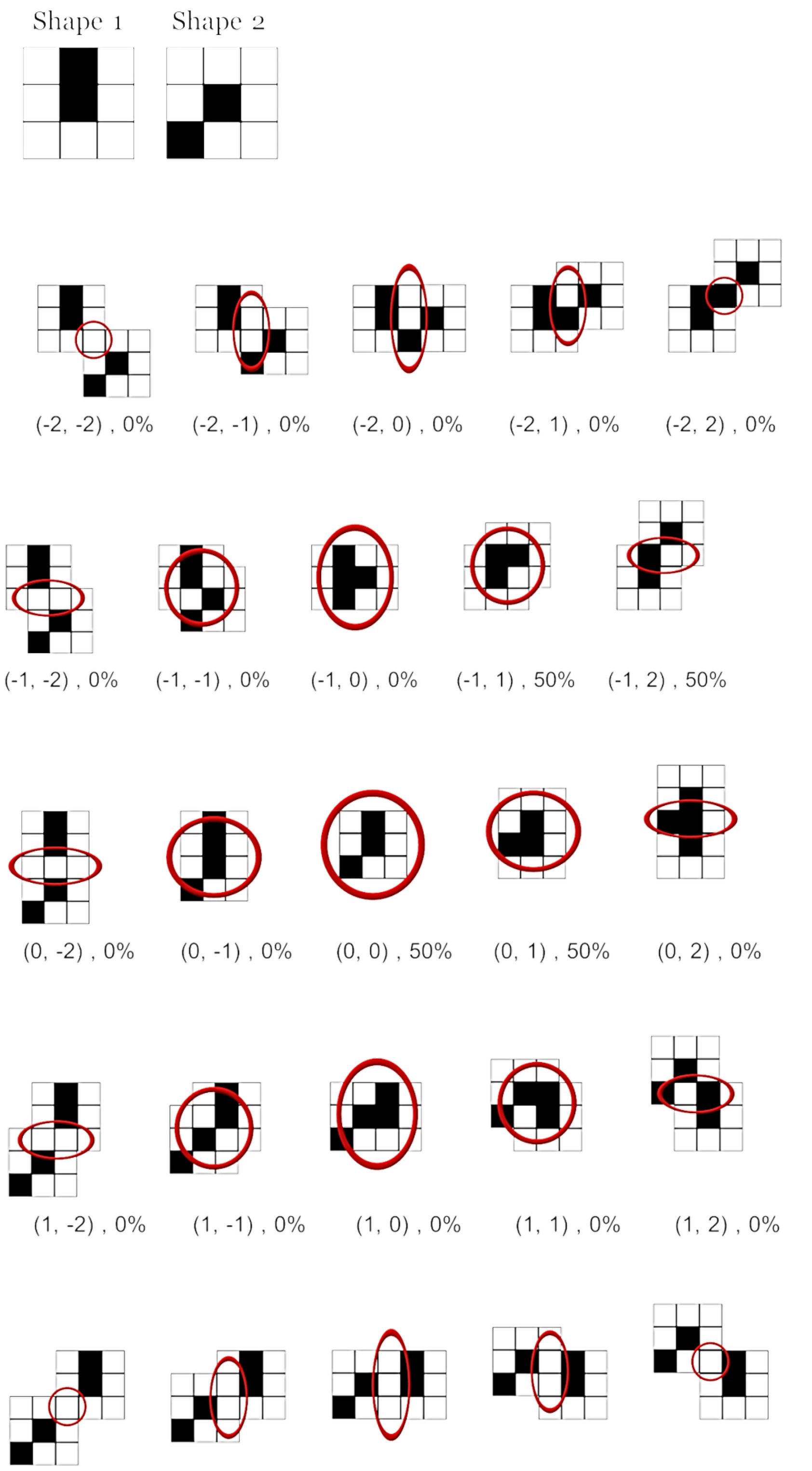


Figure 2.3 Part B. All translations that should be checked for given shapes are illustrated. The circled areas of shapes will be compared with each other. The first number inside the parentheses refers to translation units on X axis and second number refer to translation units on Y axis and the percentage value is the match percentage.

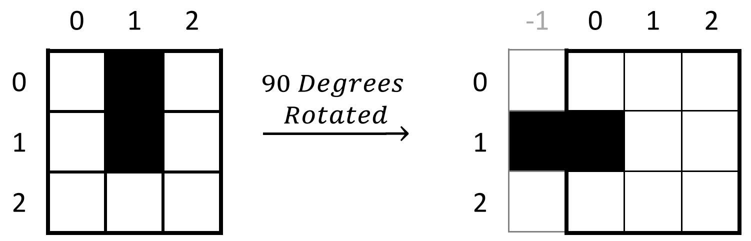
I use the rotation matrix for applying θ degrees rotation on a shape, the rotation matrix could be written using matrix multiplication as following:

$$X' = X\cos\theta - Y\sin\theta$$

$$Y' = X\sin\theta + Y\cos\theta$$

Therefore, as an example, 90 degrees rotation of Shape-1 can be calculated as following:

	X	Y	X'	Y'
Point 1	1	0	$1\cos 90 - 0\sin 90 = 0$	$1\sin 90 + 0\cos 90 = 1$
Point 2	1	1	$1\cos 90 - 1\sin 90 = -1$	$1\sin 90 + 1\cos 90 = 1$



As shown above, the new coordinate for Point-2 on X axis falls out of the matrix that presenting the Shape-1. This is not a problem or bug, because:

a. Translation process can bring this point inside the matrix.

b. This allows us to check for partial matches. As an example, for shapes in Figure 2.4 we may not be able to determine any transformation that maps them to each other using the mentioned approaches, but by benefiting the advantage of this point (i.e. allowing some sections of shape to fall out of the valid ranges of matrix) we can determine a proper transformation. If we compare the valid region of Shape 1 – RT (i.e. the point with coordinates ≥ 0) with Shape 2, then we will have $100\ \%$ match between these two shapes.

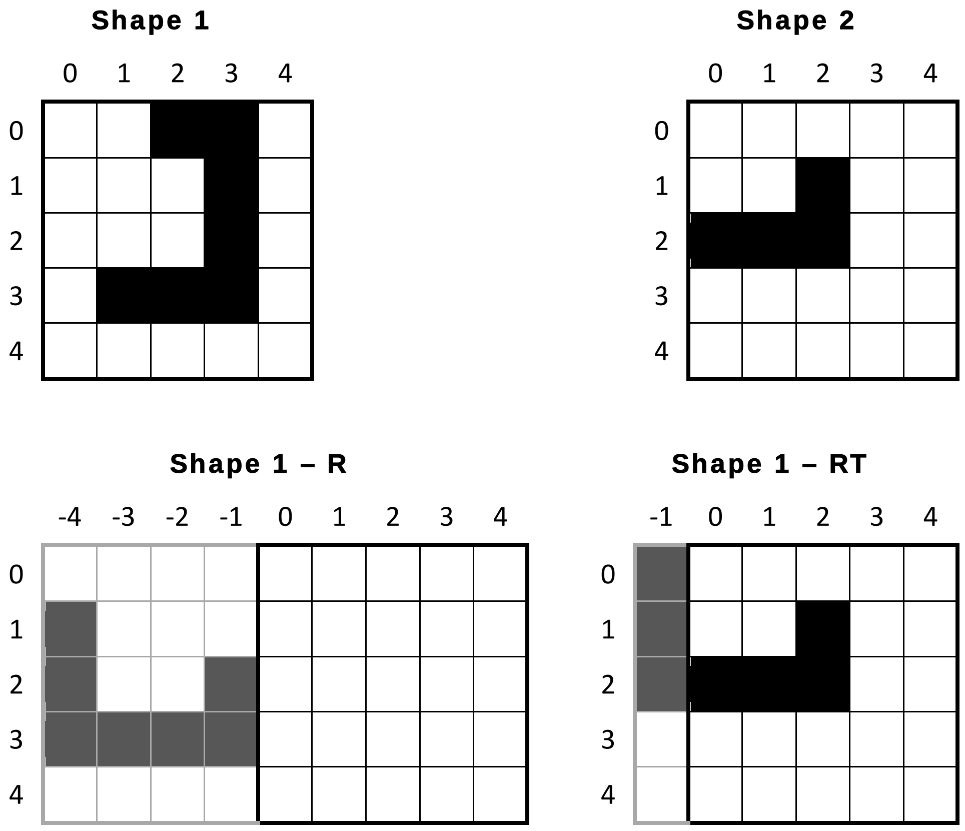


Figure 2.3 A sample of partial match between two shapes is illustrated.

Shape 1 – R: 90 Degrees rotation of Shape 1

Shape 1 – RT: Transformed (X : +3 , Y : -1) Shape 1 – R

2.2 Advantages and Disadvantages

Since this method is checking for all combinations of translation and rotation in discreet space, we can argue that this method will definitely determine the transformation between the input shapes, if and only if the transformation is a combination of translation and rotation. But unfortunately this is the only mentionable advantage of this method in contrast to some significant disadvantages. To study the disadvantages we would consider two sample shapes presented by two fairly small, 50×50 matrices (here we consider the worst case scenario, where all cells has value = 1; in practice this may happen rarely, because such a shape means nothing but a full black (or any other color) box that for obvious reasons has no value in transformation determination procedure). With following assumptions, the details of expenses of using the naïve approach are given in Table 2.1.

$$r = 360$$

$$X = (50 \times 2) - 1 = 99$$

$$Y = [(50 \times 2) - 1]^2 = 9{,}801$$

$$C = 50 \times 50 = 2{,}500$$

$$\textit{For applying Translation} \begin{cases} X_{new} = X_{old} + X \\ Y_{new} = Y_{old} + Y \end{cases}$$

Table 2.1 Expenses of the naïve approaches usage

Description		Count
Search Tree Size	Nodes	$r + rX + rY \cong 3.56 \times 10^6$
	Leaves	$rY \cong 3.52 \times 10^6$
Rotation	Sin	$2rC = 1.8 \times 10^6$
	Cos	$2rC = 1.8 \times 10^6$
	Floating Point Multiplication	$4rC = 3.6 \times 10^6$
	Floating Point Addition / Subtraction	$2rC = 1.8 \times 10^6$
Translation	Integer Addition/Subtraction	$2rCY = 1.764 \times 10^{10}$
Comparing	Compare Times	$rY = 3.52 \times 10^6$
	Two Cells Comparison	$rCY = 8.82 \times 10^9$

It is obvious from these quite large numbers (for a quite small size matrix) that this naïve approach is not practical at all. Despite of the method, it is the cardinality of the cells that we deal with which results the impracticality of this method. To decrease the number of cells I tried:

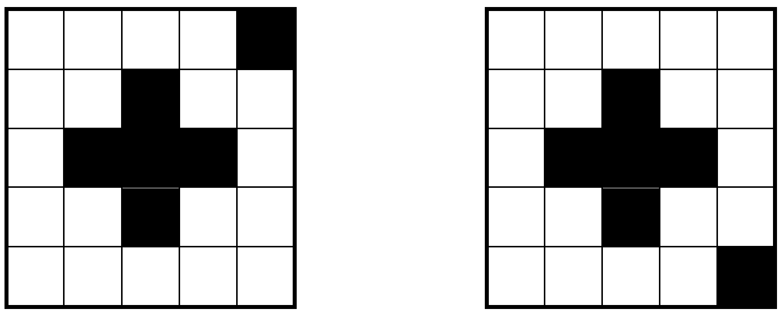
A. Random Figures

Instead of handling full shapes, we may abstract a random shape out of the initial shape which has less number of points than the initial shape. Since we are decreasing the number of points, the number of required calculations will be decreased which results increasing running speed. The accuracy of this method is in direct relation with the number of cells we choose randomly, that is, for more selected cells we will have more accurate results; but by increasing the number of randomly selected cells actually we are rolling back to the initial problem!

B. Scale – RAW

Another technique that I checked was Scale – RAW, that is, before checking for transformation I resized the shape to get a smaller shape which has fewer cells to handle. This technique could increase the running speed too, but the same problem as **Random Figures** exists here as well.

Both mentioned techniques share a considerable problem that is, in the process of random selection or rescaling, there is no guaranty that we select or keep some key points of the shapes which are critical for transformation determination procedure. As an example consider the following shapes



The only difference between these two shapes is a single cell (located on upper-right and bottom-right corners), but we cannot guaranty that we will have these cells in manipulated shapes (i.e. abstracted or scaled shapes). This weakness makes these techniques unreliable for transformation determination procedure.

2.3 Implementation

I have implemented this naïve approach in C# .NET 4.0 and here I will explain some major sections of the code briefly. As we discussed, shapes are presented by matrices in this implementation and for optimization reasons, only the incidences (the cells of matrix which has value = 1) are stored in a $n\ by\ 2$ matrix (n points and $(x\,, y)$ coordinate for each point). Shapes could be input whether manually using a GUI (Graphical User Interface) as shown in Figure 2.4 or by loading previously designed and saved shapes.

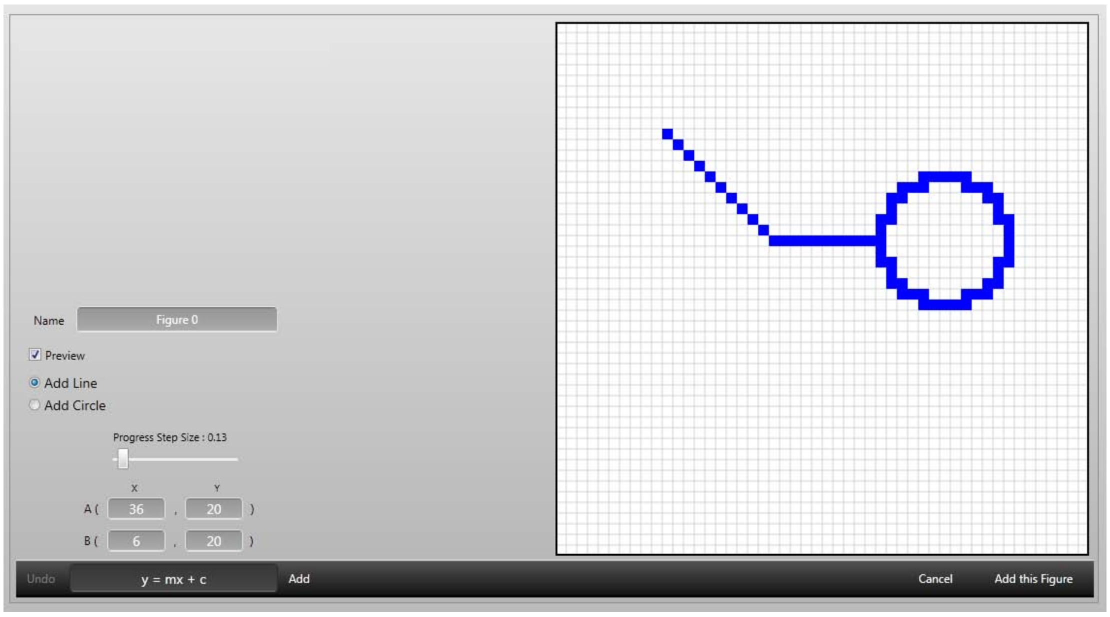

Name: Figure 0
[✓] Preview
[● Add Line]
Add Circle
Progress Step Size : 0.13
A (36 , 20)
B (6 , 20)

Figure 2.4 The manual shape input interface.

A simplified code for implementation of the structure demonstrated in Figure 2.2 is as follows:

```
void Run_Reasoning()
        {
            while (Rotate())
            {
                X_step = -X;
                Y_step = -Y - 1;

                while (Move())
                    Compare();
            }
        }
```

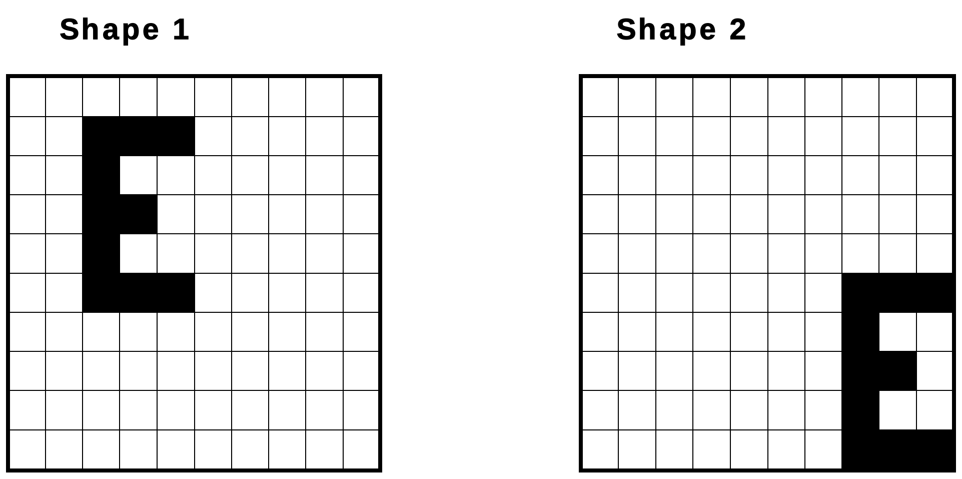
```
bool Rotate()
        {
            if (Teta < 360)
            {
                Apply_Rotation();
                return true;
            }
            else
                return false;
        }
bool Move()
        {
            Y_step++;

            if (Y_step <= Y)
            {
                Apply_Move();
                return true;
            }
            else
            {
                X_step++;

                if (X_step <= X)
                {
                    Y_step = -Y;
                    Apply_Move();
                    return true;
                }
                else
                    return false;
            }
        }
```

In this implementation I used neither *Random figures* nor *Scale – Raw* optimizations. The optimization I applied in coding affects Translation check process where significantly increased running speed and decreased the number of nodes to check for. The optimization consists of two parts, **Cropping** and **Out-pour control.** As I illustrated in Figure 2.5, most of the space of the matrices presenting the shapes are empty, therefore our search for any match/partial-match between the two shapes in Figure 2.5 in the empty regions will not give us any answers, thus cropping the shapes and searching inside the crop will be a great advancement. Crop is a rectangular region that whelms the shape; the upper-left and bottom-right corners of the crop rectangles for each of the shapes in Figure 2.5 is given under the matrices.

Although cropping optimizes the translation process by defining a new range for translation check, but still we will be checking for some translations which are far from resulting proper answers. As an example, consider the translation $(X{:}\,3\,, Y{:}\,0)$, this translation puts only the cell at coordinate $(4\,,5)$ – bottom-left cell, inside the crop of Shape-2 and leaves most of the cells of Shape-1 beyond the crop, therefore this translation could not be useful for the process (because of comparing a fraction of Shape-1 with complete shape-2). To avoid these translations, we define a parameter named **Out-pour threshold** which limits the translation ranges to the ranges those keep the percentage of Shape-1 defined by out-pour threshold inside the crop of Shape-2. This parameter can be defined either manually of allowing the application to determine it automatically.



$Crop: \; [(2\,,1)\,,(\,4\,,5\,)]$ $[(X_{11}\,,Y_{11})\,,(X_{12}\,,Y_{12})]$

$Crop: \; [(\,7\,,5\,)\,,(\,9\,,9\,)]$ $[(X_{21}\,,Y_{21})\,,(X_{22}\,,Y_{22})]$

		Range		Length
		From	*to*	
X – Axis	*Before Crop*	-9	9	19
	After Crop	$X_{21} - X_{12} = 3$	$X_{22} - X_{11} = 7$	$(7 - 3) + 1 = 5$
Y – Axis	*Before Crop*	-9	9	19
	After Crop	$Y_{21} - Y_{12} = 0$	$Y_{22} - Y_{11} = 8$	$(8 - 0) + 1 = 9$

Figure 2.5 Cropping the Shapes avoids unnecessary search for matches; the range of translation search before optimization is defined above and is compared with the new range for translation search with this optimization.

CHAPTER THREE
REASONING WITH SHAPES
SEGMENTING – LEVELING APPROACH

3.1 Introduction

If we are asked regarding to the people we saw while walking on a street, we may not be able to remember anyone, but we can’t argue that we did not see people on street. We see the entire environment within our visual range, but we only see focused objects in details and the rest of environment unclear, hence it is reasonable to argue that instinctively we make abstractions of our environment with a level of details we need at moment. As another example, consider a driver driving at high speed in a highway, the driver sees the cars in front, but he only pays attention to the distance of the cars and their speed and avoids unnecessary details such as exact model of the cars, plate numbers and drivers, in order to handle the situation. This reveals the vital role of abstractions in our life; this technique is the base of my **Segmenting – Leveling** approach.

Consider Figure 3.1; if we are asked to determine the transformation between the two shapes; we can determine landmark points[1] (Pseudo – Landmark) of the shapes and present them in two matrices, and then using Procrustes analysis or solutions of Wahba’s problem we can accurately determine the transformation between the shapes. But in some situations we may prefer to sacrifice this accuracy to achieve a fast and cheap (from aspect of complexity) approximate answers. As for Figure 3.1, we may accept the answer “*about 45 degrees Rotation*” while the accurate answer is “*33 degrees Rotation*”.

[1] A shape can be described by a finite set of points, named Landmark points. Landmark points have three different types as follows:

i. Landmark points assigned by an expert to represent a biological object that is called Anatomical Landmarks.
ii. Landmark points that are assigned by mathematical property is known as Mathematical Landmarks.
iii. Pseudo Landmark is between i and ii

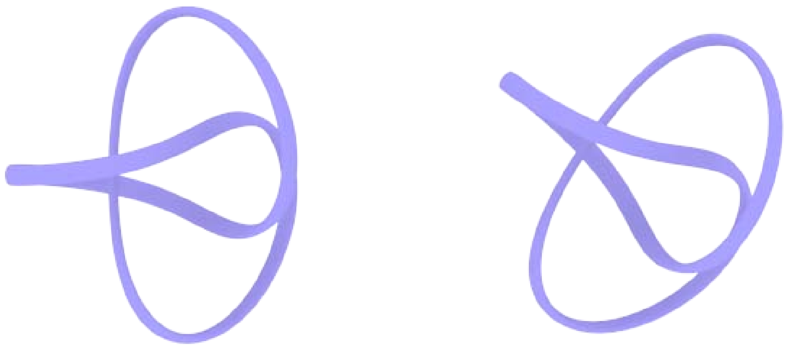


Figure 3.1 Determine the transformation between these two shapes

In Segmenting-Leveling method, like the naïve approach, I will do a state space search using a tree structure similar to the structure given in Figure 2.2. A pair of shapes will be considered, a transformation will be applied of one of the shapes and that shape will be compared with the other one, the transformations which result in best match measures, are candidate transformations. Generally:

$$Input\ shapes : Shape_1 , Shape_2 , \ldots , Shape_n$$

$$All\ possible\ transformations : T = \left\{ T_1 , T_2 , \ldots , T_j \right\}$$

$$i \in \{ 1 , 2 , \ldots , n-1 \} , t \in \{ 1 , 2 , \ldots , j \}$$

$$\forall\, i , t : C_{i\,t} = Compare\,(T_t\, Shape_i , Shape_{i+1})$$

$$\forall\, i, t\ \exists\, t' \in \{ 1 , 2 , \ldots , j \} : C_{i\,t'} \geq C_{i\,t}$$

$$Reason\ best\ transformation\ of\ the\ input\ from\ the\ set\ C_{i\,t'}$$

In contrast to naïve approach that search a discreet space for best transformations, Segmenting-Leveling approach runs in continues space. Hence, I will run a state space search more than once based on required accuracy. Also unlike naïve approach that manipulates shapes, Segmenting-Leveling approach uses abstractions of shapes. My approach runs in levels that on each level it tries to tune the results from previous level to find more accurate results; at each level I will run a state space search on a space much smaller than the space of previous level (Note that, even on initial level, we will have much smaller space to check than the space of naïve approach). In continue, I will explain this method step by step with details of each step, but for now I give a very brief visualization of this method on Figure 3.2 for determination of rotation.

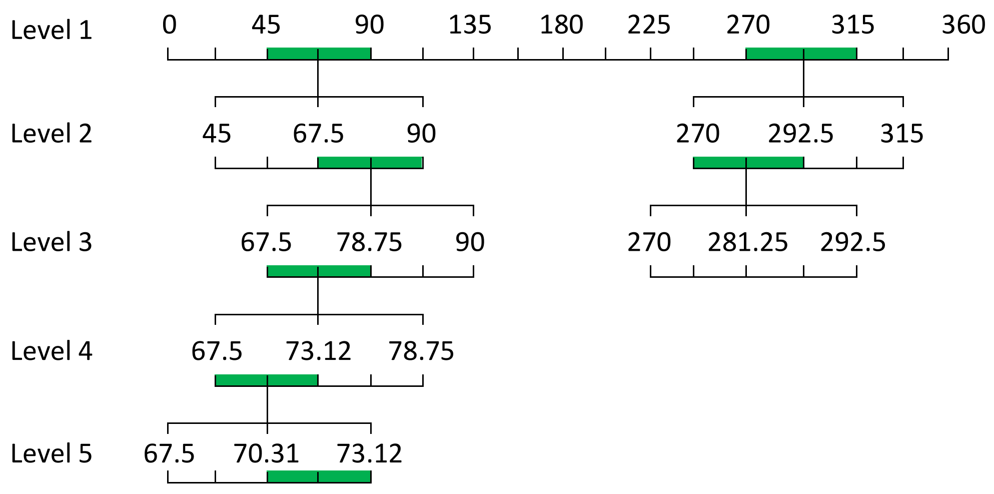


Figure 3.2 In this figure a brief visualization of rotation determination procedure in Segmenting-Leveling approach for two assumptive shapes is given. This process could be continued as many levels as required to achieve a satisfactory accurate result. Green sections at each level are the sections which the answer is estimated to be in that range, hence, I break that range into two identical ranges and again I continue to estimate on which range the answer could be, and this process will be continued until a tolerable accurate answer is found. For this example the answer: 70.31 – 73.12 (range) is a tolerable answer and searching process stops at this point. Note that, the searching in the range 270 – 292.5 is stopped, because distance measurement process did not mark either of the ranges as a proper range.

3.2 Segmentation

I divide the shapes presented in 2 dimensional Euclidian space into N isometric triangular *Regions* and each *Region* is divided into M identical *Segments*, where *Regions* are denoted by Euclidian unit vectors V_n , $n \in \{ 1 , 2 , \dots , N \}$, *Segments* are denoted by vectors V_{nm} , $m \in \{ 1 , 2, \dots , M \}$ and an angle θ and cardinality of landmark points inside each segment which is presented by S_{nm} . The center of *Segmentation* is considered to be the center of the rectangle that environs the shape. This process is demonstrated in Figure 3.3.

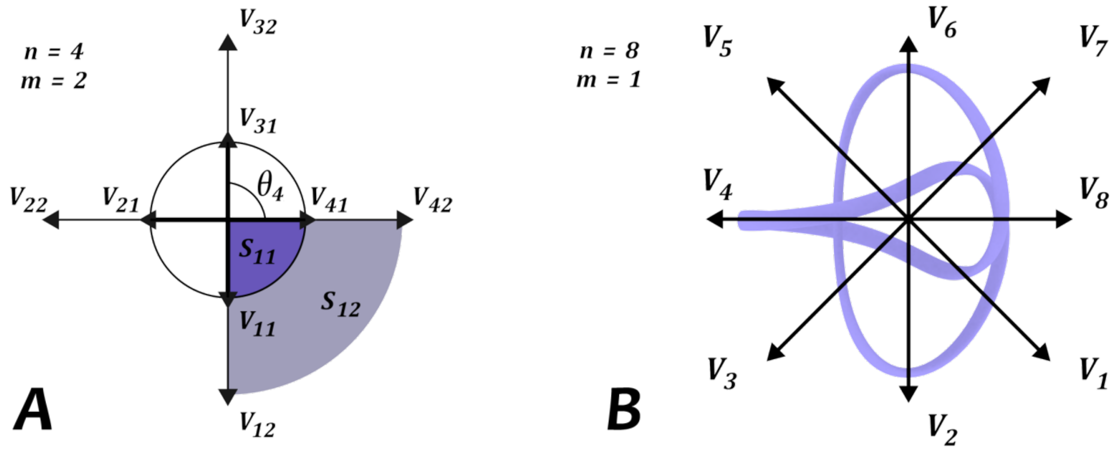


Figure 3.3 (A) Each figure will be divided into N isometric Regions and each Region will be divided into M identical Segments, the Segments are denoted by V_{nm} vectors. The angle between each pair of vectors is denoted by θ_n where $\forall\, n \in \{1, 2, \ldots, n\} \rightarrow \theta_{n-1} = \theta_n$. Cardinality of landmark points inside each segment is denoted by S_{nm}.

(B) As a sample segmentations vectors are shown on one of the shapes from Figure 3.1.

We have the following matrix presenting the segmentation vectors:

	1	2	…	M
Region 1	V_{11}	V_{12}	…	V_{1M}
Region 2	V_{21}	V_{22}	…	V_{2M}
…	…	…	…	…
Region N	V_{N1}	V_{N2}	…	V_{NM}

Each of these vectors is presented by 2 coordinates to which I add S_{nm} as 3rd dimension as following:

V_{nm}	x	y	S_{nm}

This process is visualized in Figure 3.4 for a hypothetical initial shape.

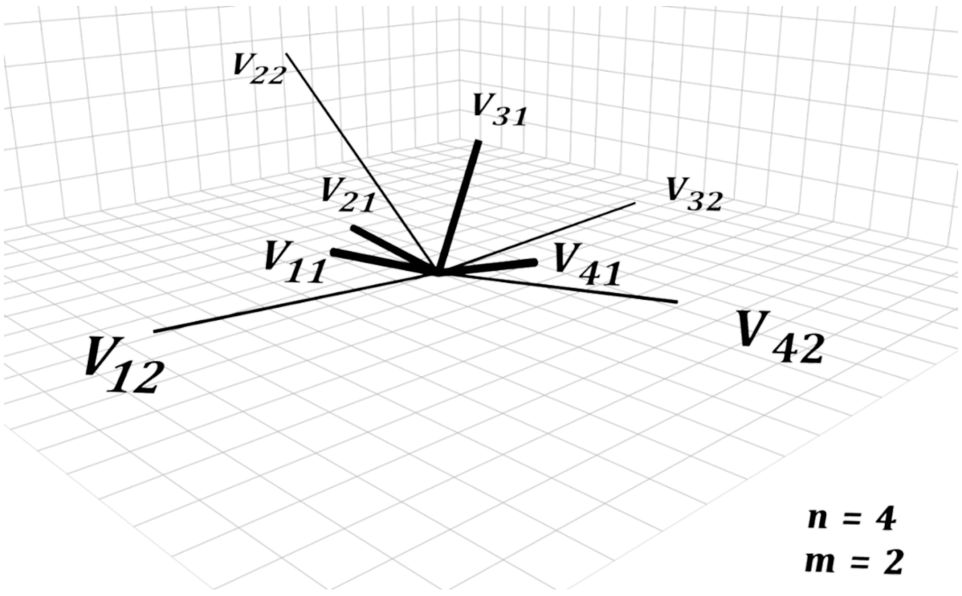


Figure 3.4 Visualization of adding $\mathbf{S_{nm}}$ as **3**rd dimension to 2-Dimensional segmentation vectors of a hypothetical initial shape is given.

Throughout my reasoning method, I only need the last dimension of each vector, that is S_{nm} , and the order of vectors for the method's processes, hence I rewrite the segmentation vectors matrix as following and name it as A.

A :

	1	2	…	M
Region 1	S_{11}	S_{12}	…	S_{1M}
Region 2	S_{21}	S_{22}	…	S_{2M}
…	…	…	…	…
Region N	S_{N1}	S_{N1}	…	S_{NM}

The order of vectors can tell us the approximate coordinate of each vector, but needless of knowing the exact coordinate of each vector is an advantage, because then we don't have to translate shapes to a specific coordinate to be able to check for other transformations; meaning that, we can do linear transformation determination process in place. Although we don't need to know even the approximate coordinate of each vector, but consider the following procedure as an example for determination of approximate coordinate of each vector in 2-Dimmensional space:

$$\theta = \frac{360}{N} \quad , \quad r = \frac{1}{M}$$

$$(because\ by\ definition\ a\ unit\ vector\ presents\ a\ Region$$

$$and\ each\ Region\ is\ devided\ into\ M\ segments\ )$$

$$\sin\theta = \frac{y}{r} \rightarrow y = r\sin\theta$$

$$\cos\theta = \frac{x}{r} \rightarrow x = r\cos\theta$$

Each shape may have Υ landmark points presented in 2 dimensional space as following matrix, since we choose M, N as $\Upsilon \gg MN$, therefore the matrix A will be significantly smaller than the matrix presenting the landmark points.

	x	y
Landmark Point 1	P_{1x}	P_{1y}
Landmark Point 2	P_{2x}	P_{2y}
…		
Landmark Point Υ	$P_{\Upsilon x}$	$P_{\Upsilon y}$

Segmentation process gives us a none-unique abstraction of each shape (i.e. different shapes could have same abstractions – I will cover this point in section 3.7). The important point is: ***this abstraction gives us a matrix regardless of the size of shapes and number of landmark points*** (i.e. no matter if we have billions of landmark points presenting the shape or only a few, for all sizes of shapes we have $N \times M$ integers presenting the shape). With this advantage, the cost of reasoning huge shapes will be as cheap as the cost of small shapes.

Through the rest of this chapter, we consider the shapes in 2 dimensional space for the sake of simplifying explanation and understanding of the method, although because we only need the $(d+1)^{th}$ dimension of a d dimensional shape, the generalization of this method for d dimensional shapes is possible and fairly simple.

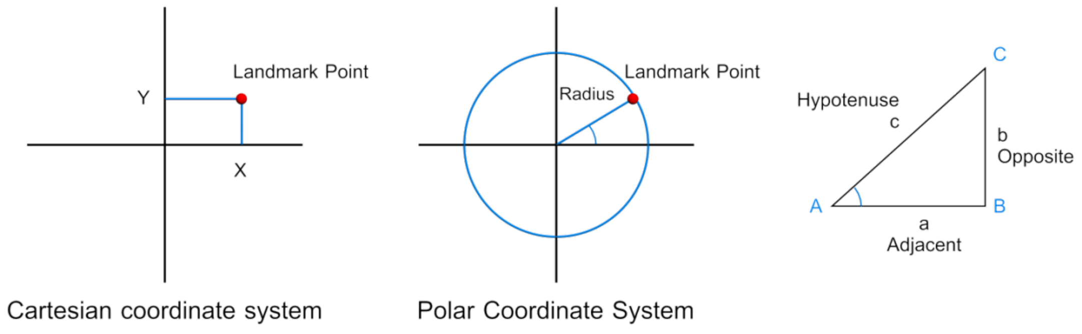


Figure 3.5 Different coordinate systems that might be used to coordinate landmark points are illustrate

3.3 Segment Determination for a Landmark Point

For segmentation purpose, we should be able to determine which segment a landmark point belongs to? The answer for this question is directly dependent to the coordinate system (Figure 3.5) which we use to coordinate landmark points. Proper segment detection in two coordinate systems, Polar and Cartesian, are explained below, note that, although the segments might be presented using Spherical or Cylindrical coordinate systems, but for the purpose of complexity, I focus on two common coordinate systems which are Cartesian and Polar coordinate systems.

3.3.1 Polar Coordinate System

I start with Polar coordinate system because it is similar to the nature of my segmentation. We consider *Pole* to be coincided with center of segmentation and *Polar Axis* coincided with floor of first region. Now we can determine the segment of a landmark point P distinguished by a *Polar angle* θ and *Radius (Radial Coordinate)* r as following:

$$\left\langle \exists n \in \{1, 2, \dots, N\} \;\middle|\; \frac{360}{N}(n-1) < \theta \leq \frac{360}{N} n \right\rangle \qquad (3.3.1.1)$$

$$\left\langle \exists\, m \in \{\, 1, 2, \dots, M \,\} \;\middle|\; \frac{1}{M}(m-1) < r \le \frac{1}{M}\, m \right\rangle \qquad (3.3.1.2)$$

Hence, the segment distinguished by $(\, r, \theta \,)$ in polar coordinate systems belongs to segment $(\, n,\ m \,)$.

3.3.2 Cartesian Coordinate System

We consider the origin of coordinate system and the X axis to be coincided with center of segmentation and floor of first region respectively. Since the regions and segments are based on circular divisions, therefore it could be easier to first convert a Cartesian coordinate to Polar coordinate and then use the ranges mentioned in *Polar coordinate system* section to determine the segment which the landmark point belongs to.

$$P :\ (\, x, y \,) \qquad (3.3.2.1)$$

$$Radius : r^2 = x^2 + y^2 \rightarrow r = \left|\, \sqrt{x^2 + y^2} \,\right| \qquad (3.3.2.2)$$

$$Reminder :\ \sin\theta = \frac{b}{c}\ ,\ \ \cos\theta = \frac{a}{c}\ ,\ \ \tan\theta = \frac{a}{b}\ \ (see\ Figure\ 4.5) \qquad (3.3.2.3)$$

$$\rightarrow\ \ \theta = \sin^{-1}\frac{y}{r}\quad , \theta = \cos^{-1}\frac{x}{r}\quad , \theta = \tan^{-1}\frac{x}{y} \qquad (3.3.2.4)$$

$$\rightarrow\ \ \left\langle \exists\, m \in \{\, 1, 2, \dots, M \,\} \;\middle|\; \frac{1}{M}(m-1) < \left|\, \sqrt{x^2 + y^2} \,\right| \le \frac{1}{M}\, m \right\rangle \qquad (3.3.2.5)$$

Either of the followings

$$\left\{ \begin{array}{l} \left\langle \exists\, n \in \{\, 1, 2, \dots, N \,\} \middle| \dfrac{360}{N}(n-1) < \tan^{-1}\dfrac{x}{y} \le \dfrac{360}{N} n \right\rangle \\[2ex] \left\langle \exists\, n \in \{\, 1, 2, \dots, N \,\} \middle| \dfrac{360}{N}(n-1) < \sin^{-1}\dfrac{y}{r} \le \dfrac{360}{N} n \right\rangle \\[2ex] \left\langle \exists\, n \in \{\, 1, 2, \dots, N \,\} \middle| \dfrac{360}{N}(n-1) < \cos^{-1}\dfrac{x}{r} \le \dfrac{360}{N} n \right\rangle \end{array} \right. \qquad (3.3.2.6)$$

Accordingly, the landmark point $P\ (x, y)$ belongs to the segment $n\,, m$.

3.4 Translation

Translation of all shapes to a common coordinate so that the centroid of all shapes coincide, is a prerequisite for the linear transformation determination process in previously mentioned approaches. Since I make abstraction of shapes by segmentation and as I mentioned in *Segmentation* section, the nature of the Segmenting–Leveling method makes translation operation for the reason of mentioned approaches unnecessary for my method.

I define an auxiliary application for translation in my method, and that is for checking the partial matches between shapes, which is an advantage of my method over mentioned methods that can’t determine any partial matches. An example of partial match is given in Figure 2.3.

I would define translation process between two shapes as moving the segmentation center of one shape to coordinates pointed out by the segmentation vectors of the other shape. Three sample cases of this process are illustrated in Figure 3.6 for two abstractions of two assumptive shapes with $N = 8$ and $M = 1$.

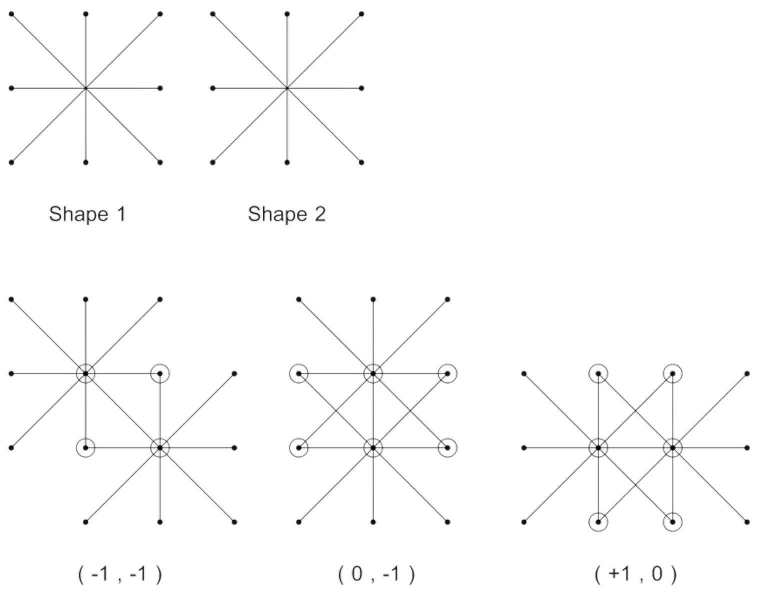


Figure 3.6 Three sample cases of translation process are illustrated. The circled areas shows the vectors which should be compared with each other. (-1,-1) , (0,-1) , (+1,0) are the transformation parameters.

To generalize the translation task, I would take the following steps:

1. Define sets T_x and T_y as following:

$$T_x = \left\{ \left(\frac{-1}{m}\ ,0\ ,\frac{1}{m} \right) \middle|\ m = 1\,,2\,,\dots,M \right\}$$
$$T_y = \left\{ \left(\frac{-1}{m}\ ,0\ ,\frac{1}{m} \right) \middle|\ m = 1\,,2\,,\dots,M \right\} \qquad (3.4.1)$$

2. Define set T as the combination of sets T_x and T_y
3. Check for all members of T.

The last step of generalization is similar to the translation check procedure in naïve approach except in the number of required checks. In naïve approach the number of checks is directly related to the size of shapes that for larger shapes more checks are required than smaller shapes, but in Segmenting–Leveling approach, the maximum number of checks is $(NM + 1)$ (i.e. moving to all coordinates pointed out by segmentation vectors of the other shape plus no translation) which according to definition, is much fewer than the number of checks in naïve approach.

3.5 Rotation

Rotation is an isometric circular transformation of a rigid body around a pivot – unlike translation that has not any fixed point – on a plane or space. Mainly two different types of rotation are defined, Spin and Revolution (Orbital Revolution), which Spin is a rotation with the pivot inside the mass of the rigid body and Revolution is a rotation with pivot outside the rigid body. In geometry Revolution is also defined as Spin + Translation, which is spinning the object around any pivot and then translating the object so that the pivot of spinning coincides with the pivot of requested Revolution. Rotation on a plan can be carried out using the following matrix, known as rotation matrix.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Using matrix multiplication we have the following equations for determination of new coordinates (x', y') of (x, y) with θ degrees rotation.

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

Regarding to sample rotations in Figure 3.7 we can make the following arguments:

Section A . 90 degrees rotation swaps the position of colors one unit, for example 4 takes the position of 1 while 1 takes the position of 2 and so on, but keeps the location of 0 unchanged. Hence we can argue that 180 degrees rotation swaps the position of colors two unit and 270 degrees swaps the position of colors three units. Generally we can say: any $90\,i$ degrees of rotation, swaps the position of colors i units. Although we can use rotation matrix to determine new position of each color with any θ degrees rotation, but this generalization helps us to guess the new position of each color in a much easier way. Simply this generalization is not useful for rotation degrees other than $90\,i$, and leaves rotation matrix as our only choice.

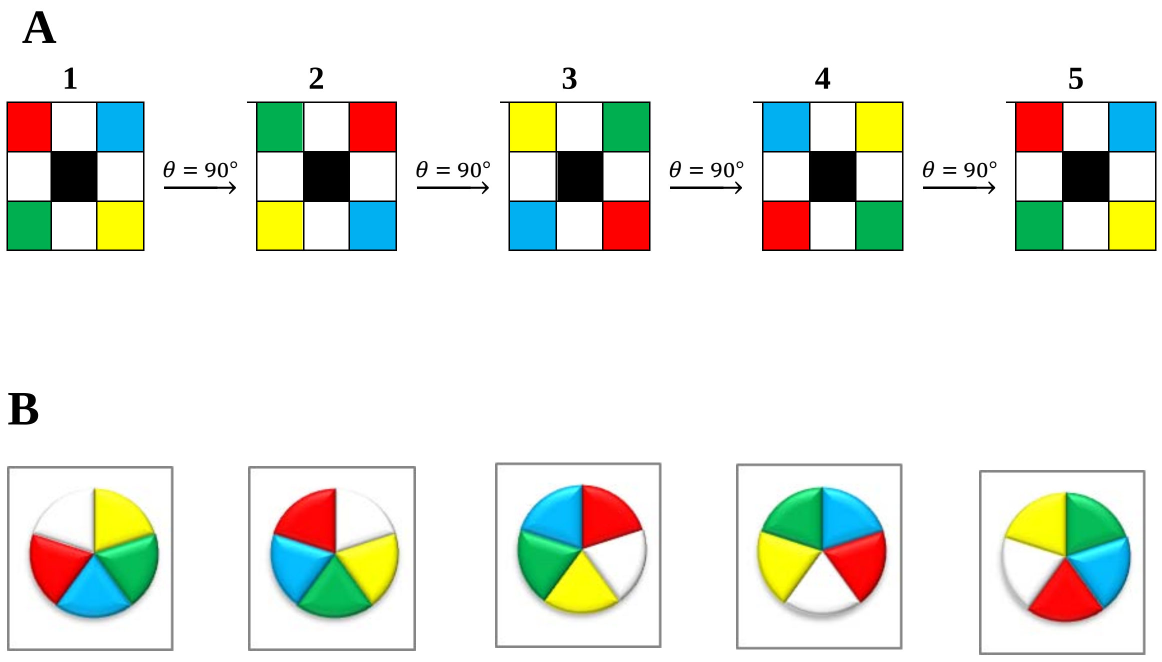


Figure 3.7 Sample rotations

Section B . With 72 degrees rotation slices change position one unit, as an example, 72 degrees clockwise rotation moves:

- **White** slice to the position of **Yellow** slice,
- **Yellow** slice to the position of **Green** slice,
- **Green** slice to the position of **Blue** slice,
- **Blue** slice to the position of **Red** slice and
- **Red** slice to the position of **White** slice.

Hence we can argue (as **Section A**) for any $72\,i$ degrees rotation, slices **Shift** i units.

Accordingly we may reason: *If a shape is divided into* i *identical slices then* $(\,360\,/\,i\,)\,j\ ,\ \ j \in \{\ 1\ ,2\ ,\dots,\ i-1\ \}$ *degrees rotation is the same as* j *units* **shifting** *slices.*

One of my reasons of defining Segmenting–Leveling approach is to use simplest possible operations for reasoning with shapes, hence, although rotation matrix that I mentioned early in this section can cover my needs for rotation determination process, but **Shift** is a much simpler operation than trigonometry functions, which I prefer to use it**.** But as I mentioned, we can use **Shift** instead of only a few number of rotations, therefore using the definition of segmentation I would define a set of rotations which I can use **Shift** to manipulate them and I restrict the rotations that my method can determine to only the members of this set. The set is as following:

$$R = \left\{ \frac{360}{N}\, i \;\middle|\; i = 0\,,1\,,2\,,\dots\,,N-1 \right\}$$

$$e.g.\ \ N = 8 \ \ \Rightarrow R = \{\ 0°\ ,45°\ ,90°\ ,135°\ ,180°\ ,225°\ ,270°\ ,315°\ \}$$

Now I can use a **Circular Shift** on matrix A (the matrix defined in section **4.2**) to rotate my figure $\theta\,°\ ,\theta \in R$. As an example:

$$\mathrm{N} = 6\,,\mathrm{M} = 2$$

$$\mathrm{R} = \{\,0\,,60\,,120\,,180\,,240\,,300\,\}$$

$$\begin{array}{c|c|c|} & 1 & 2 \\ \hline \text{Region 1} & S_{11} & S_{12} \\ \hline \text{Region 2} & S_{21} & S_{22} \\ \hline \text{Region 3} & S_{31} & S_{32} \\ \hline \text{Region 4} & S_{41} & S_{42} \\ \hline \text{Region 5} & S_{51} & S_{52} \\ \hline \text{Region 6} & S_{61} & S_{62} \\ \hline \end{array} \xrightarrow{\substack{\text{Rotate:120 Degrees} \\ = \\ \text{2 time circular shift}}} \begin{array}{c|c|c|} & 1 & 2 \\ \hline \text{Region 5} & S_{51} & S_{52} \\ \hline \text{Region 6} & S_{61} & S_{62} \\ \hline \text{Region 1} & S_{11} & S_{12} \\ \hline \text{Region 2} & S_{21} & S_{22} \\ \hline \text{Region 3} & S_{31} & S_{32} \\ \hline \text{Region 4} & S_{41} & S_{42} \\ \hline \end{array}$$

Clockwise and counterclockwise rotations have the same procedure, but we have to agree on one and use it throughout whole procedures, here I choose clockwise rotations, therefore everywhere on this thesis when I mention rotation, it means clockwise rotation.

A child can rotate an object without having any knowledge of geometrical definition of rotation and trigonometry; therefore I tried to define a method for rotating a shape much similar to the way a child might use than the normal methods which are benefiting from rotation matrix and trigonometry functions. My defined method uses **Circular Shift** that is much simpler and cheaper than trigonometry functions. My proposed method can rotate a shape $\theta\,°\;,\theta \in R$ using only circular shift, but limiting rotation degrees to a finite set of angles is a significant inefficiency and I will cover this up by making some edits on this method in Leveling section (3.7) which enables my method to check for all rotations degrees in continues space rather than current discreet space.

3.6 Match Measurement

I measure match ratio between two shapes where one of the shapes is RAW (i.e. no transformation is applied on it) and the other one is the transformed shape.

$$Match\ Ratio = Compare\ (\ TShape_A\ , Shape_B\ )$$

I divide match measurement task into two separate steps:

i. **Count** the number of segments of $TShape_A$ that match with corresponding segments in $Shape_B$.

ii. Determine **match ratio**

Consider the following abstractions of two assumptive shapes:

$Shape_A$

	1	2
1	0	0
2	0	0
3	0	0
4	1	0
5	46	163
6	0	0

$\xrightarrow{T{:}R\ (\ 120°\ )}$

$T\ Shape_A$

	1	2
5	46	163
6	0	0
1	0	0
2	0	0
3	0	0
4	1	0

$Shape_B$

	1	2
1	46	163
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0

To **Count** the matching segments, we could easily compare each row of $T\ Shape_A$ with corresponding row of $Shape_B$. If we do so, all rows will match except the last rows that the 1st segment of 4th region of $Shape_A$ do not match with 1st segment of 6th region of $Shape_B$, but the 2nd segments of these regions match. Hence we could say 11 segments out of 12 segments of these abstractions are matched with the applied transformation.

If the difference between two shapes is a member of set TR (combination of Translation (T) and Rotation (R)) then with this method of comparing two shapes, we might be able to determine a match, but if the difference between these shapes is not a member of set TR then this way of comparing two shapes would not be useful. For example consider the abstraction of two assumptive shapes given in Figure 3.8

where the difference between two shapes is 115 degrees rotation that is not a member of R:

$$R = \{0, 15, 30, 45, 60, 75, 90, \ldots, 345\}$$

Hence with no swiping (circular shift) we might be able to map $Shape_A$ on $Shape_B$ (shapes given by Figure 3.8). Although as I mentioned earlier using leveling technique we will be able to check for rotations in a continues space with any accuracy required at the point that could also cover 115 degrees, but as I mentioned leveling technique **tunes** the results in continues space, meaning that we should be able to estimate that the answer is about 120 degrees and then expect to determine 115 degrees in leveling process. To address this problem I used a threshold value while comparing the values of segments as below:

$$if\ \ Shape\_A\_Segment_{\ i} + threshold \le Shape\_B\_Segment_{\ i}$$

$$and\ Shape\_A\_Segment_i - threshold \ge Shape\_B\_Segment_i$$

If this condition is satisfied, then the segments will be considered as identical, if otherwise then the segments are not equal.

In my implementation (I will explain this implementation in Chapter Five) I set this value manually and I allowed the user to change it. Through some tests, I noticed that in some cases lower values of threshold are useful while higher values are preferred for some other cases.

The count of identical segments alone may not be proper factor of evaluating the similarity between two shapes. I defined a factor and named **Match Ratio,** also I defined five different functions for its calculation. The functions are as follows (J (Joint segments) is the count of segments that are considered as identical, A , TA and B are the number of segments with values greater than 0 of $Shape_A$, Transformed $Shape_A$ and $Shape_B$ respectively):

1. Find $Shape_A$ in $Shape_B$ where $Shape_A$ is not allowed to lose any of its portions.

$$Match\ Ratio = \frac{J}{A} \times 100$$

2. Find $Shape_A$ in $Shape_B$ where $Shape_A$ is allowed to lose some of its portions (e.g. Figure 2.3).

$$Match\ Ratio = \frac{J}{TA} \times 100$$

3. Find $Shape_B$ in $Shape_A$

$$Match\ Ratio = \frac{J}{B} \times 100$$

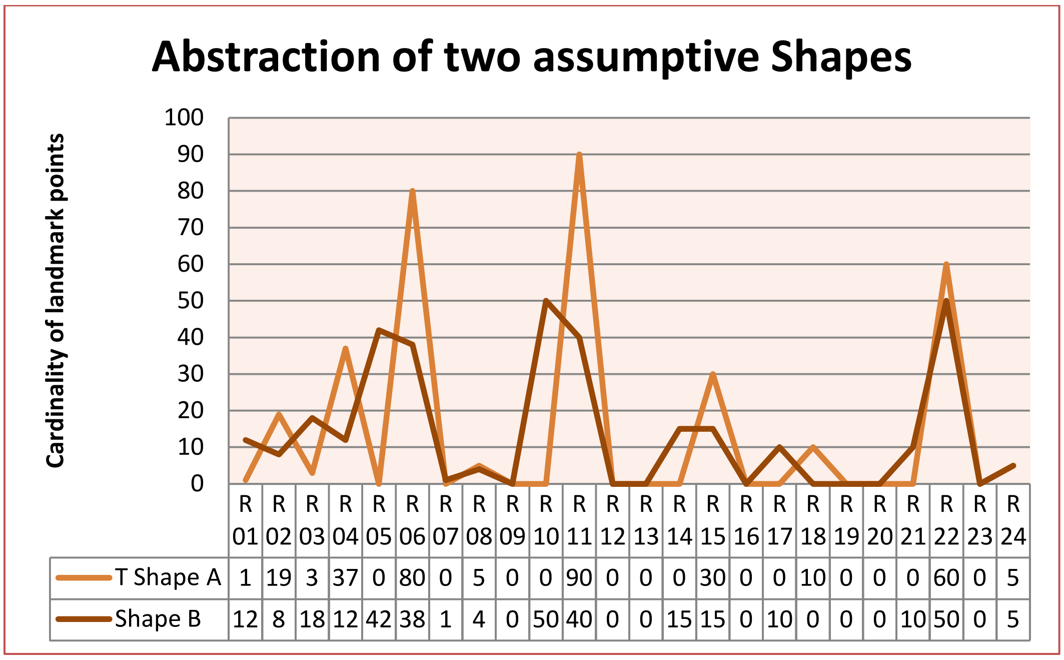
4. Compare the two shapes where $Shape_A$ is not allowed to lose any of its portions.

$$Match\ Ratio = \frac{2J}{A + B} \times 100$$

5. Compare the two shapes where $Shape_A$ is allowed to lose some of its portions.

$$Match\ Ratio = \frac{2J}{TA + B} \times 100$$

These functions are used separately and they have different applications as mentioned. In my implementation user can choose either of these to be used during reasoning process. The combination of counting method and one of the different functions of match ratio calculation gives a proper and reliable value for match measurement, although in some cases I had to tune threshold value to achieve a proper answer.



	R 01	R 02	R 03	R 04	R 05	R 06	R 07	R 08	R 09	R 10	R 11	R 12	R 13	R 14	R 15	R 16	R 17	R 18	R 19	R 20	R 21	R 22	R 23	R 24
T Shape A	1	19	3	37	0	80	0	5	0	0	90	0	0	0	30	0	0	10	0	0	0	60	0	5
Shape B	12	8	18	12	42	38	1	4	0	50	40	0	0	15	15	0	10	0	0	0	10	50	0	5

Figure 3.8 Cardinality of landmark points inside each segment of two assumptive shapes where $\mathbf{n} = \mathbf{24}$ and $\mathbf{m}$ could have any values because Regions (R) values (which are the sum of all segments inside each region) are shown here.

3.7 Leveling

Leveling is a supplement of Segmenting; it repeats segmenting procedure with different parameters until it achieves desired accurate results. Our goal is to determine a transformation $f = TR$ that maps $Shape_A$ on $Shape_B$, but as we discussed in segmenting section for both translation and rotation we have a finite set transformations as follows:

$$R = \left\{ \frac{360}{N} i \;\middle|\; i = 0, 1, 2, \dots, N-1 \right\} \tag{3.7.1}$$

$$T_x = \left\{ \left(\frac{-1}{m}, 0, \frac{1}{m} \right) \;\middle|\; m = 1, 2, \dots, M \right\} \tag{3.7.2}$$

$$T_y = \left\{ \left(\frac{-1}{m}, 0, \frac{1}{m} \right) \;\middle|\; m = 1, 2, \dots, M \right\} \tag{3.7.3}$$

$$T = Combination\ of\ T_x\ and\ T_y \tag{3.7.4}$$

As it is obvious from these sets, the accuracy of using segmentation procedure only for once is quite dependent on segmenting parameters N and M. But we cannot simply increase these parameters to achieve more accurate results, because doing so we will face a very huge search tree with a large number of segments to deal with, which is not optimal and is not practical to determine accurate transformations such as $f = 90.012°$. Leveling technique solves this problem by running segmentation procedure for many times but each time it tunes the results of previous run. For leveling purpose I define:

$$Level : L = \mathbb{N} - \{0, 1, 2\} \tag{3.7.5}$$

I have excluded $\{0, 1, 2\}$ because they would not result in proper θ for segmentation, I would generalize the definition of θ , N and M as follows:

$$\forall\, l \in L :\ N = 2^l \ ,\ M = 2^l$$

$$\theta = \frac{360}{N} \ ,\ \theta_l = \frac{1}{2}\,\theta_{l-1} \tag{3.7.6}$$

As I mentioned earlier translation is not a compulsive operation and I am using it to determine partial matches if any exists, hence if partial matches determination is not desired we can simply set $M = 1$.

I divide the leveling procedure into three phases:

Phase 1 . Initial Level

i. Choose an initial value for l; it is better to choose it not too large for optimization reasons. I start with $l = 3$ in my implementation.

ii. Run segmentation on both shapes with parameters regarding to l and make matrix A for both shapes.

iii. Apply all possible transformation on $Shape_A$ and compare it with $Shape_B$.

iv. Choose Υ best transformations (i.e. the transformations which resulted best match ratios) and pass them to next level.

Phase 2 . Non-Initial Levels

i. Increment l one unit.

ii. Run segmentation on both shapes with parameters regarding to l and make matrix A for both shapes. Incrementing l one unit, doubles the number of segments of each level than the previous level, in other words, we are dividing each segment of previous level into two identical segments for current level.

iii. For each of the transformations passed from previous level, α , check for $\alpha \pm \beta , \beta = \{ 0 ,1 ,2 , \ldots , \lambda \}$. λ Defines a range to be checked for tuning purpose and in my implementation is given manually before reasoning process is started.

iv. Choose Υ best transformations,

- If the transformations are accurate enough then continue with next pair starting at **Phase 1**
- If the transformations accuracy is not satisfactory continue to next level.
- If all pairs are processed continue to **Phase 3**.

Phase 3 . Use δ best transformations of each pair to reason the best transformation of the sequence. It is likely to have a transformation that resulted in high match measurement in one pair and a low match measurement on another pair; hence I choose transformations which have acceptable match measures for all pairs while they are not necessarily the best transformations of all pairs. Then I apply the chosen transformation on the last shape, which results the best candidate to stand as the shape after the last one in the sequence.

An example covering main portion of leveling procedure is given as following:

Level 1 . (Initial Level)

$$l = 3 \;\rightarrow\; N_1 = 8 \tag{3.7.7}$$

$$\rightarrow\; R_1 = \left\{ \frac{360}{N_1}\, i \;\middle|\; i = \{0, 1, 2, \dots, 7\} \right\} \tag{3.7.8}$$

$$\alpha_1 = \{\alpha_{11}, \alpha_{12}, \dots, \alpha_{1\Upsilon}\}$$

$$a\ set\ of\ \Upsilon\ best\ transformations\ from\ Level\ 1\ to\ pass\ to\ Level\ 2 \tag{3.7.9}$$

Level 2 .

$$l = 4 \rightarrow N_2 = 16 \tag{3.7.10}$$

$$\rightarrow\; R_2 = \left\{ \alpha_{1j} \pm \frac{360}{N_2}\, i \;\middle|\; i = \{0, 1, \dots, \lambda\} \;, j = \{1, 2, \dots, \Upsilon\} \right\} \tag{3.7.11}$$

$$\alpha_2 = \{\alpha_{21}, \alpha_{22}, \dots, \alpha_{2\Upsilon}\} \tag{3.7.12}$$

Level 3 .

$$l = 5 \rightarrow N_3 = 32 \tag{3.7.13}$$

$$\rightarrow\; R_3 = \left\{ \alpha_{2j} \pm \frac{360}{N_3}\, i \;\middle|\; i = \{0, 1, \dots, \lambda\} \;, j = \{1, 2, \dots, \Upsilon\} \right\} \tag{3.7.14}$$

I should remind that during reasoning process I don’t use any rotation angles, instead I use shifting as I explained in **Rotation** section. The R sets defined in previous example defined using rotation angles to ease the understanding of the procedure for reader, otherwise as an example, R_1 and R_2 are actually defined and used as follows:

$$R_1 = \left\{ i\,CS \,\middle|\, i = \{ 0 ,1 , \dots ,7 \}\right\} \;,\; CS : Circular\ Shift \tag{3.7.15}$$

$$\alpha_1 = \{ \alpha_{11} ,\alpha_{12} , \dots ,\alpha_{1\Upsilon} \} \tag{3.7.16}$$

$$R_2 = \left\{ 2\alpha_{1j} \pm i\,CS \,\middle|\, i = \{ 0 ,1 , \dots ,\lambda \} \;, j = \{ 1 ,2 , \dots ,\Upsilon \}\right\} \tag{3.7.17}$$

In R_2 , α is multiplied by two, because as we are dividing each segment into two identical segments for new level, we should multiply by two a circular shift on one level to address the same rotation for next level.

As I mentioned previously, segmentation might result in two identical abstracts for two different shapes. This is a likely condition and is simply addressed by leveling technique. Two shapes will have two equal abstractions in all levels (i.e. with different segmentation parameters) if and only if the two shapes are exactly the same, but if the shapes are not the same and segmentation resulted in identical abstractions then we will have proper and different abstraction in one or a few levels (i.e. by changing segmentation parameters). Because actually by running Segmenting–Leveling method for more levels, we are taking more details of shapes into consideration which finally will reveal the difference between two shapes by resulting different abstractions.

3.8 Correctness Verification

Previously on this chapter I claimed that this method can determine any transformation (any combination of Rotation and Translation) between multiple shapes and can create the shape which suits best to stand after the last shape in the sequence, here I will verify this assertion.

If the difference between two shapes is rotation and the rotation angle, $\theta \in R_1$ then obviously will be able to determine it, because as I explained in **Rotation** section, shifting will cover it up. Our problem is when $\theta \notin R_1$, and I claimed that this will be covered up by using Leveling technique. For this claim, consider the following assumption:

$$\left\langle \exists i \in \{1, 2, \dots, N\} \;\middle|\; \frac{360}{N}(i-1) < \theta \leq \frac{360}{N} i \right\rangle \qquad (3.8.1)$$

By definition of segmentation, these are the boundaries of i^{th} region that I divided it into M segments. This assumption shows that we have only one range (region) where θ resides in, therefore as much as we narrow down this range, we get closer and closer to an accurate θ. Also it is likely to confront some shapes which with multiple θs they can be mapped on each other, determination of multiple θs is also possible by Segmenting–Leveling method, because as I mentioned, at each level I consider Υ rotation angles (see Figure 3.2)(Rotations angle is a part of a set of transformations passed to next level).

Determination of a proper i is quite vital, because improper i can guide us to a dead end by eliminates proper ranges which using leveling we might be able to determine the accurate θ. To be able to choose proper i's the following solutions might be handy:

1. Threshold

I used a threshold value while comparing a pair, although choosing proper values for the threshold is a challenging problem, but since user can change it manually in my implementation we can assume by tuning it we could achieve proper values. Even though it is not reliable but for my current purpose I accept it with tunings.

2. Multi Initial Levels

Another solution which I suggest is to use multi initial levels, meaning that running a full search on possible ranges (like Level 1 as I explained) for more than one level, although this could be useful but it can significantly increase the complexity of the solution.

I claimed that using leveling that narrows down the (3.8.1) range; we could have an acceptable narrow range with θ considered to be rounded to the upper bound of the range, for this allegation consider the following:

$$by\ definition : l \in \mathbb{N} - \{0, 1, 2\}, \qquad N = 2^l \tag{3.8.2}$$

$$\begin{gathered} Level\ 1 : l = 3 \rightarrow N = 8 \\ Level\ 2 : l = 4 \rightarrow N = 16 \\ Level\ 3 : l = 5 \rightarrow N = 32 \\ \ldots \\ \Longrightarrow \lim_{l \rightarrow \infty} 2^l = \infty \rightarrow N = \infty \end{gathered} \tag{3.8.3}$$

$$\frac{360}{\infty}(i-1) < \theta \leq \frac{360}{\infty} i \quad \rightarrow \quad 0 < \theta \leq 0 \tag{3.8.4}$$

This shows that we can continue leveling procedure for infinitely many times and at last we will have a range as narrow as it overlaps with θ. Here are some points:

a. As we noticed, i is not important in infinity, therefore continuing to infinity could not be useful, although it is not practical.

b. Actually we don't need to continue leveling procedure to infinity, the number of levels we should run to achieve our desired accuracy can be calculated as following:

$$Accuracy : \varepsilon \tag{3.8.5}$$

$$The\ range\ of\ segmentation\ should\ be\ as\ narrow\ as\ \varepsilon^{\circ}$$

$$:\ \frac{360}{N}\,i - \frac{360}{N}\,(i-1) = \varepsilon \qquad (3.8.6)$$

$$\frac{360\,i - 360\,i + 360}{N} = \varepsilon \rightarrow \frac{360}{N} = \varepsilon \qquad (3.8.7)$$

$$N = 2^{l} \rightarrow \frac{360}{\varepsilon} = 2^{l} \rightarrow l = \log_2 \frac{360}{\varepsilon} \qquad (3.8.8)$$

$$\rightarrow l = \left\lceil \log_2 \frac{360}{\varepsilon} \right\rceil \qquad (3.8.9)$$

Hence we can determine the rotation angles between two shapes by running finite number of levels as calculated above, note that, the calculated l is the maximum number of levels we need, however it is possible to achieve accurate results even in more early levels, this happens when θ is the border at a level or very close to the border. I concentrate on Rotation because the main goal of transformation determination in most of the previous works was Rotation, but however Translation is also verifiable in this way.

CHAPTER FOUR
IMPLEMENTATION
SEGMENTING – LEVELING APPROACH

In this chapter, I will explain a program that I developed to examine the Segmenting–Leveling method using Visual Studio .NET 4.0 technology; for this implementation, Windows Presentation Foundation (WPF) is used for interfaces and code behind is written in C#. To ease the tracing, I limited the number of levels to four; however the results are still acceptable.

The implementation could be recursive, but since accessing statistical data (used for purpose of studying the performance) of each iteration is much simpler when the implementation is not recursive, therefore I defined separate classes for each level. Note that, non-recursive implementation would not be possible if I would not have had restricted the number of levels to four.

The main function that runs levels between a pair has following procedures:

1. Define classes Level – 1, Level – 2, Level – 3 and Level – 4. Despite of a lot of functions and parameters, each class has a member function named "Start_Reasoning" that runs reasoning process at the level and a list named "Matches" that stores best match measurements of transformation determination process results.
2. Initialize classes with options that can be changed and tuned by user (see Figures in Appendix for the screenshot of this section)
3. Call the "Start_Reasoing" function of Level – 1.
4. Pass best transformations resulted from Level – 1 to Level – 2 (refer to section 3.7 for details).
5. Call the "Start_Reasoning" function of Level – 2.
6. Pass best transformations resulted from Level – 2 to Level – 3.

7. Call the “Start_Reasoning” function of Level – 3.
8. Pass best transformations resulted from Level – 3 to Level – 4.
9. Call the “Start_Reasoning” function of Level – 4.
10. Store the best transformations resulted from Level – 4 to be used in logic determination process between the input sequences of shapes.

Since all of the levels have same structure, therefore in continue I will explain the structure and procedures taken in Level – 1 as a sample level. The steps are as follows:

1. Segmentation; make matrix A (see section 3.2) for each of the shapes of given pair. Since the initial shapes (2-Dimensional) are presented by matrices (2-Dimensional), therefore in linear time ($\theta\ (xy)$ where x is the number of columns and y is the number of rows) we could read the initial shapes and create the matrix – A – liked matrix for each of shapes.

2. Check for transformations using segmentation matrices. This procedure can be done using a structure similar to the simplified code given in section 2.3.

3. When each transformation is applied on $Figure_A$ it will be compared with $Figure_B$ and results are stored in a list of list of arrays defined and initialized as follows in C#:

```
List<List<int[]>> Matches = new List<List<int[]>>();

for (int i = 0; i < 101; i++)
{
    List<int[]> t = new List<int[]>();
    Matches.Add(t);
}
```

Now if we consider the match ratio (see section 3.6) is denoted by “Match_R”, then this transformation is stored as following in “Matches”:

```
Matches[Match_R].Add(new int[] { Rotation_Angle, TX, TY });
```

Where TX and TY are translation parameters on X and Y axis respectively (see section 3.4). This method of storing checked transformations, guarantees sorted transformations regarding to their “Match Ratio” at each step and eases the selection of transformations with desired value of match measurement, hence the complexity of sorting is constant and equals to $\theta\,(1)$ also the complexity of fetching transformations with desired value of match measurement is constant as well.

4. The only difference between the simplified code given in section 2.3 and Segmenting–Leveling approaches code is the domain for each of “Apply_Rotation” and “Apply_Move” functions, where the domain of “Apply_Rotation” function is quite dependent to the best transformations passed from previous level and the domain of “Apply_Move” function depends on the pruning techniques explained in section 2.3. In creating the domain of “Apply_Rotation” function, duplicate rotations are avoided. Rotations from a level are passed to next level using following code (as an example here, from Level – 1 (L1) to Level – 2 (L2)):

```
int n = 0; // Number of Added Transformations
for (int i = 100; i >= 0; i--)
{
   for (int j = 0; j < L1.Matches[i].Count; j++)
   {
      // R_to_C : Rotations to be Checked
      if (!L2.R_to_C.contains(L1.Matches[i][j][0]))
      {
         n++;
         if (n <= Gama)
            L2.R_to_C.Add(L1.Matches[i][j][0]);
         else
         {
            j = L1.Matches[i].Count;
            i = 0;
         }
      }
   }
}
```

R_to_C is defined in each level as following:

```
List<int> R_to_C = new List<int>();
```

And finally the domain for “Apply_Rotation” that is denoted by “R” (see section 3.7) is defined and initialized as follows:

```
List<int> R = new List<int>();

private void initialize_Rotations()
{
   // Note that R_to_C.count = Gama
   for (int i = 0; i < R_to_C.Count; i++)
      R.Add(R_to_C[i]);


   for (int i = 0; i < R_to_C.Count; i++)
   {
      for (int j = -Upsilon; j < 0; j++)
      {
         int t = R_to_C[i] + j;

         if (!R.Contains(t))
            R.Add(t);
      }

      for (int j = 1; j < Upsilon + 1; j++)
      {
         int t = R_to_C[i] + j;

         if (!R.Contains(t))
            R.Add(t);
      }
   }
}
```

The reason that rotations passed from previous level are added to R separately and at first is that, because if we would have same match measurements for “`R_to_C[i]`” and “`R_to_C[i] + Upsilon`” we prefer to consider the “`R_to_C[i]`” rotation rather than “`R_to_C[i] + Upsilon`” for tunings and reasoning process, which this will not be possible (because of the structure of the implementation) unless we insert the “`R_to_C[i]`” to the beginning of the R list.

The R set creation process runs in constant time that is $\theta\ (Gama + 2\ Upsilon)$; because Gama and Upsilon are values set before the process is started and are independent of the size of input shapes.

Now that the domains for “Apply_Rotation” and “Apply_Move” functions are defined the runtime complexity of transformation check procedure could be calculated as following:

$$O\,((\gamma + 2\,\Upsilon)\ M^2)$$

$$\omega\,(\gamma + 2\,\Upsilon\ )$$

Please refer to section 3.4 for M . Best case runtime (ω) is achievable when the pruning techniques for Translation determination section works perfect or when partial matches are not desired and are avoided. Worst case runtime (O) is expected when partial matches are desired and pruning techniques cannot prune even one translation. Note that, the runtime complexity is independent of the size of input shapes and is dependent to some parameters set by user where with the changes I mentioned in Chapter Four, these dependencies can be reduced.

Having finished running levels, best results of transformation determination process between each pair are determined and stored sorted regarding to their score value in an array named “Overal_Result”. A sample of a portion of this array is given in Table 5.1, and the generation process of this array is given below:

```
int[,] Overall_Results = new int[360, 3];

for (int i = 0; i < 101; i++)
{
    for (int j = 0; j < L1.Matches[i].Count; j++)
    {
        if (Overall_Results[L1.Matches[i][j][0], 0] < i)
            Overall_Results[L1.Matches[i][j][0], 0] = i;

        Overall_Results[L1.Matches[i][j][0], 1] =
            Overall_Results[L1.Matches[i][j][0], 1] + i;

        Overall_Results[L1.Matches[i][j][0], 2]++;
    }
}
```

The given code is for Level – 1 only, which the same code is used for all checked levels as well. Using this process we determine the best match measurement achieved by a rotation angle, the number of times we met the rotation angle in "Matches" list and the score of that rotation angle which is the sum of all match measures we have for that specific rotation angle.

Table 4.1 A sample of determined results of transformation between a pair

Score	Rotation	Best Match	Occur count
4400	8	100 %	171
3257	22	95 %	85
2879	1	92 %	67

A similar procedure as the last code will be used to determine best logics between the pairs (instead of reading from "Matches", results from "Overall_Results" of all pairs will be considered), and applying these logics on the last shape standing in the input sequence, we can determine the best and all candidates to be the next standing in the sequence.

I mainly concentrated on transformation determination process of this implementation and I tried to program it as optimal as possible. The analysis of runtime complexity shows a constant value regarding to the parameters set be user, which reveal a fair complexity of algorithm and implementation. Furthermore, the analysis of code and algorithm shows that this algorithm has an end.

In this chapter I tried to cover all major points of the implementation, although there are many points and techniques used in this implementation both for sake of increasing running speed and algorithm optimization which are not mentioned here. Different types of data structures and programming techniques in addition to many features of XAML visual effects used to illustrate results are being benefited in this implementation. Some screenshots of the implementation are given in Appendix.

CHAPTER FIVE
CONCLUSION AND FURTHER RESEARCH

The optimization I applied in coding of the naïve approach (Chapter Two) quite depends on input shapes, meaning that, for some shapes it works well while on some other shapes it has less effect. For the shapes that their crop size is almost the same as matrix size or even close to it, the optimization methods would not be handy. The other problem with this approach is the nature of its search pattern that it does not take into consideration the match measurement at each step, better say, it does a blind search. Consider the Figure 4.1; this chart is a part of the monitoring the performance of this method on a sample test.

As is obvious in the Figure 4.1, at some stages this method approaches some proper match values, but instead of tuning the parameters to get better results, it continues to new parameters which in some stages led to worst match values. Although finally it determines the best transformation but this transformation might be achievable via much less transformation checks.

I have tested the implementation of naïve approach on a number of shapes; overall average results of tests are presented in Table 4.1. This naïve approach may have proper answers in acceptable period for transformation determination process between a set of shapes, but since the optimization techniques are not reliable, this technique is not trustworthy.

Table 5.1 the average results my implementation's application on some test shapes.

		Rotations	Translation		Total	Optimization Ratio
			X - Axis	Y - Axis		
No Optimization		360	99	99	3,564,360	0%
With Optimization	Simple Shape	360	7	14	35,280	99%
	Normal Shape	360	28	34	342,720	90%
	Complicated Shape	360	41	57	841,320	76%

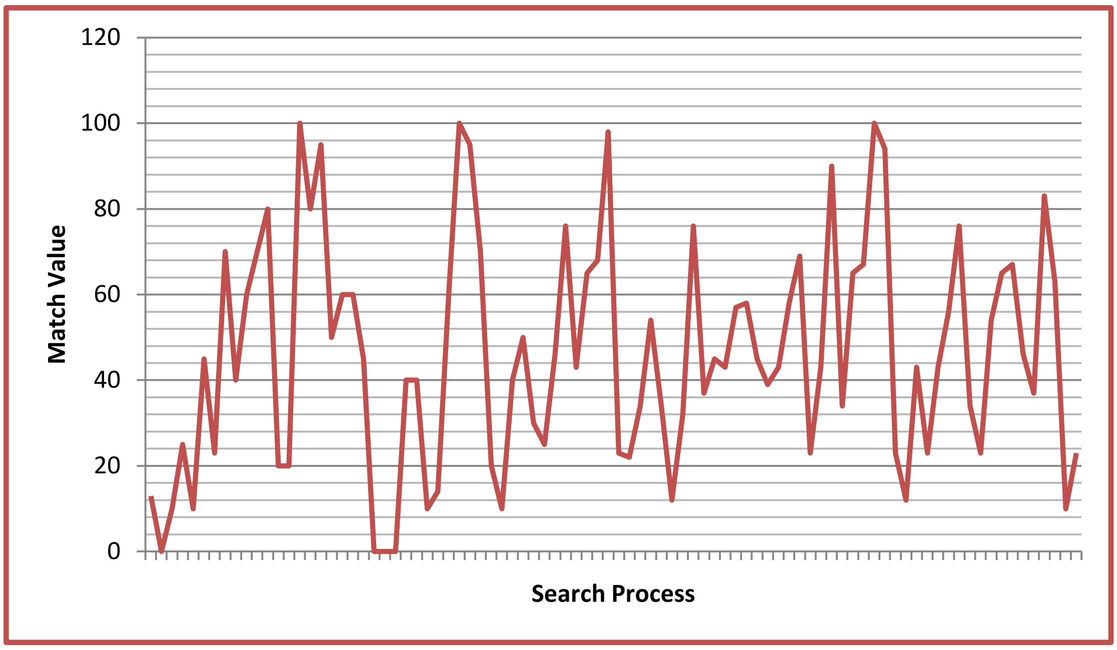


Figure 5.1 Monitoring of transformation determination process in naïve approach is given, and shows that the naïve approach runs a state space search without taking into consideration the match measurements of each compare.

In Chapter Three I proposed a solution for determination of best candidates to stand after the last member of a sequence of shapes and I named it Segmenting–Leveling method. Furthermore, because I am using basic aspects of shapes for reasoning, I would also call this method “**Reasoning with Shapes”**. The first task for the reasoning purpose is to find the logic of the sequence, for this I tried to compare sequential shapes in pairs and determine the logic between them, then using these logics, I tried to reason the logic of the set and applying the logic on the last shape I deduced the candidate shape.

To compare two shapes, I propose to derive abstractions of shapes and manipulate them instead of actual shapes. To make abstractions, I divide shapes into segments and set the cardinality of landmark points inside each segment as the value of the segment. Then I try to guess the transformation by checking for all possible transformations for the abstraction, and using Leveling I try to tune the guessed transformations to achieve a satisfactory accurate results.

I verified the correctness of my proposed method; I showed that we will be able to determine the transformations and guess the requested shape, and also I proved that using finite and acceptable number of levels we can achieve as accurate results as we need.

This method has numerous advantages over the naïve approach that I explained in Chapter Two. One of the most important advantages is the abstraction technique which this method benefits from that makes the runtime and complexity of this method independent of the size of shapes. The other advantage is, searching in continues space rather than discreet space that the naïve approach uses, which enables Segmenting–Leveling method to be able to determine transformations with any accuracy required that naïve approach is incapable of. Another advantage is complexity, where my proposed method can do transformation determination with much less compares and operations than the naïve approach.

My method has some advantages over the methods available in literature. One of these advantages, as over naïve approach, is segmentation which makes the complexity of Segmenting–Leveling method independent of the size and complexity of input shapes. Another advantage is the complexity of operations, where the methods in literature use some complex roots such as SVD based roots which are costly to be used for simple operations with limited resources where my method use Circular Shift as basic operation which is much cheaper and easier to manipulate rather than SVD based roots. Also available methods have to deal with complex and costly matrix manipulations such as matrix multiplications that could be time and resource consuming when applied on huge matrices, where my proposed method is free of this problem.

This method suffers from a proper definition for threshold value; it seemed to me that a dynamic threshold value might be a better option than a static value that I used here, which requires further research.

As I mentioned one of the main goals of my method is to use as simple and least resources as possible, although most of the method’s procedure satisfies this requirement, but segmentation task uses trigonometry functions to determine a proper segment for each landmark point which is not as simple as shift operation used for rotation determination. Since the main application of this method is in robotics, we may be able to use robots vision system to do segmenting by changing focus point and multi sensors similar to compound eyes of insects (in contrast to non-compound eyes such as human eyes) to do segmenting task. This could be a great achievement that requires further research.

REFERENCES

Alsina, C. (2009). *When Less is More: Visualizing Basic Inequalities.* The Mathematical Association of America publications.

Bruzzone, L., & Callegari, M. (2010). *Application of the Rotation Matrix Natural Invariants to Impedance Control of Rotational Parallel Robots.* Hindawi Publishing Corporation Advances in Mechanical Engineering.

Choi, Y. (2011). Efficient shape modeling using occupancy reasoning with reconstruction scheduling for interactive virtual environments. *VR Innovation (ISVRI), IEEE International Symposium,* 287-291.

Davenport, P. D. (1968). A Vector Approach to the Algebra of Rotations with Applications. *Tech. rep., NASA TN D-4696, Greenbelt, MD.*

Devlin, K. (2000). *The Language of Mathematics: Making the Invisible Visible.* Holt Paperbacks publications.

Fallon, L., & Sturch, C. R. (1979). Ground Attitude Determination and Gyro Calibration Procedures for the HEAO Missions. *Proceedings, AIAA 17th Aerospace Sciences Meeting, New Orleans, Louisiana.*

Gover, J., & Dijksterhuis, G. (2004). *Procrustes Problems.* OXFORD university press.

Guo, B., & Vachharajani, N. (2007). Shape analysis with inductive recursion synthesis. *Proceedings of the 2007 PLDI conference,* 42 (6), 256-265.

Kabsch, W. (1976). A solution of the best rotation to relate two sets of vectors. *Acta Crystallographica,* 922-923.

Kabsch, W. (1978). A discussion of the solution for the best rotation to relate two sets of vectors, *Acta Crystallographica,* 827-828.

Markley, F. L., & Mortari, M. (2000). Quaternion Attitude Estimation Using Vector Measurements, *The Journal of the Astronautical Sciences, 48,* 359-380.

Markley, F. L. (1988). Attitude determination using vector observations and the singular value decomposition. *The Journal of the Astronautical Sciences, 36,* 245-258.

Markley, F. L. (1993). Attitude Determination Using Vector Observations: a Fast Optimal Matrix Algorithm. *The Journal of the Astronautical Sciences, 41,* 261-280.

Markley, F. L., & Mortari, M. (2000). Quaternion Attitude Estimation Using Vector Measurements. *The Journal of the Astronautical Sciences, 48,* 359-380.

Mortari, D. (1996). EULER–2 and EULER–n Algorithms for Attitude Determination from Vector Observations. *Space Technology, 16,* 317-321.

Mortari, D. (1997). *ESOQ:* A Closed–Form Solution to the Wahba Problem. *Journal of the Astronautical Sciences, 45,* 195-204.

Mortari, D. (2000). Second Estimator of the Optimal Quaternion. *Journal of Guidance, Control and Dynamics, 23,* 885-888.

Nelsen, B. (1993). *Proofs without Words: Exercises in Visual Thinking.* The Mathematical Association of America publications.

Nelsen, B. (2001). *Proofs Without Words II: More Exercises in Visual Thinking.* The Mathematical Association of America publications.

Nelsen, B. (2006). *Math Made Visual: Creating Images for Understanding Mathematics.* The Mathematical Association of America publications.

Polster, B. (2004). Q.E.D.: *Beauty in Mathematical Proof.* Walker & Company publications.

Reynolds, R. G. (1998). Quaternion parameterization and a simple algorithm for global attitude estimation. *Journal of guidance, control and dynamics. 21,* 669-671.

Schuster, M. D., & Oh, S. D. (1981). Three-Axis Attitude determination from vector observation. *Journal of guidance and control, 4,* 70-77.

Schuster, M. D. (1978). Approximate Algorithms for Fast Optimal Attitude Computation. *AIAA Guidance and Control Conference, Palo Alto, California,* 88-95.

Shuster, M. D., & Natanson, G. (1993). Quaternion computation from a geometric point of view. *Journal of Astronomical science, 41,* 545-556.

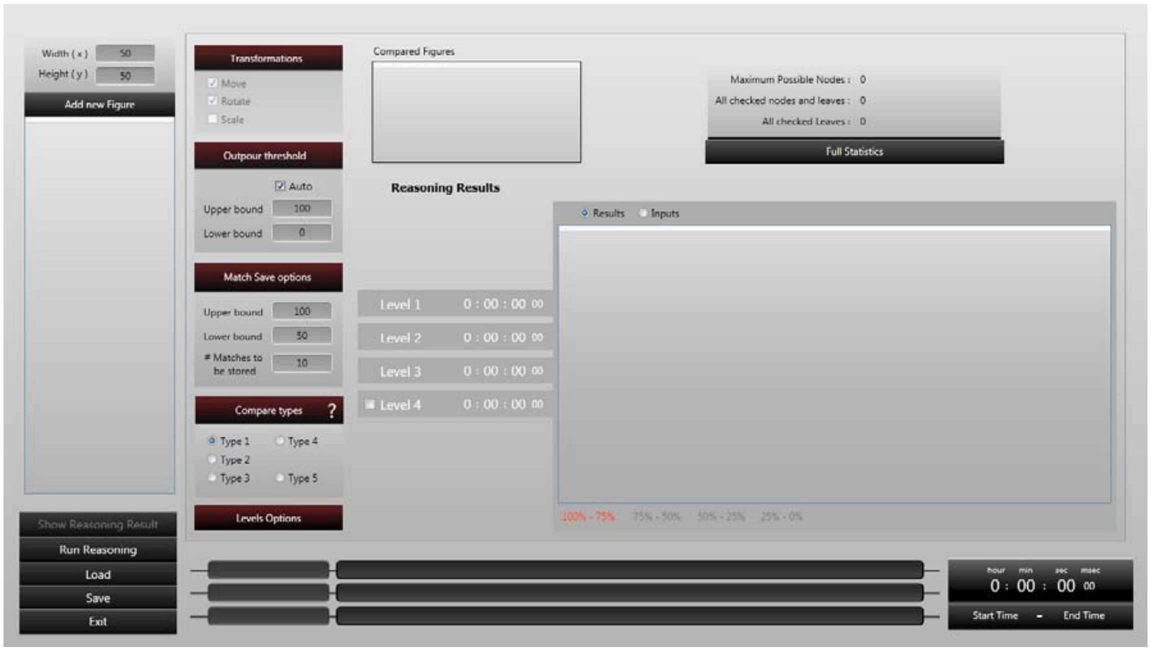
Stark, L. (1996). Recognizing object function through reasoning about partial shape descriptions and dynamic physical properties. *IEEE Journal,* 1640-1656.

Thomas, V. (2006). *Algorithms for the Weighted Orthogonal Procrustes Problem and other Least Squares Problems*, Ph.D. Thesis, UMINF-06.10, UMEA University.

Wahba, G. (1965). A Least Squares Estimate of Spacecraft Attitude. *SIAM Review*, 7, 409, Problem 65-1.

APPENDIX

Main window is shown on Figure A.1; here user can set the size of the matrices which shapes are presented by, the Outpour Threshold value for all levels and choose either of compare types. Also user can see the results of each level presented separately and grouped as 100% - 75%, 75% - 50%, 50% - 25% and 25% - 0%. User can choose whether to see the results of each level or the transformations passed to next level for tuning purpose.



Width (x) 50
Height (y) 50

Transformations: [✓ Move] [✓ Rotate] Scale

Outpour threshold: [✓ Auto]
Upper bound 100
Lower bound 0

Match Save options:
Upper bound 100
Lower bound 50
Matches to be stored 10

Compare types: [✓ Type 1] Type 2 Type 3 Type 4 Type 5

Maximum Possible Nodes : 0
All checked nodes and leaves : 0
All checked Leaves : 0

Level 1 0 : 00 : 00 00
Level 2 0 : 00 : 00 00
Level 3 0 : 00 : 00 00
Level 4 0 : 00 : 00 00

[✓ Results] Inputs

Figure A.1 Main Window

User may choose whether to Load previously saved shapes by clicking on Load button or manually add shapes by clicking on Add New Figure button. If saved shapes are loaded, matrices sizes are read from the saved file and set and will not changeable. While manually defining shapes, matrices sizes must be set using two text boxes on upper-right corner of the window before clicking on Add new figure button.

Manually adding figure form is shown on Figure A.2, user should try to draw shapes by using the combination of Lines and Circles. Defined shapes are listed on the left most of the form and by double clicking on each of them user can review the drawn shape at any time; an example of this is given in Figure A.3.

By clicking of Levels Options button, a set of options will be shown that allows us to set some properties for each of the levels such as the excepted range of match ratios to be passed to next level, maximum number of transformations to pass to next level and matching threshold value. These options are shown in Figure A.4.

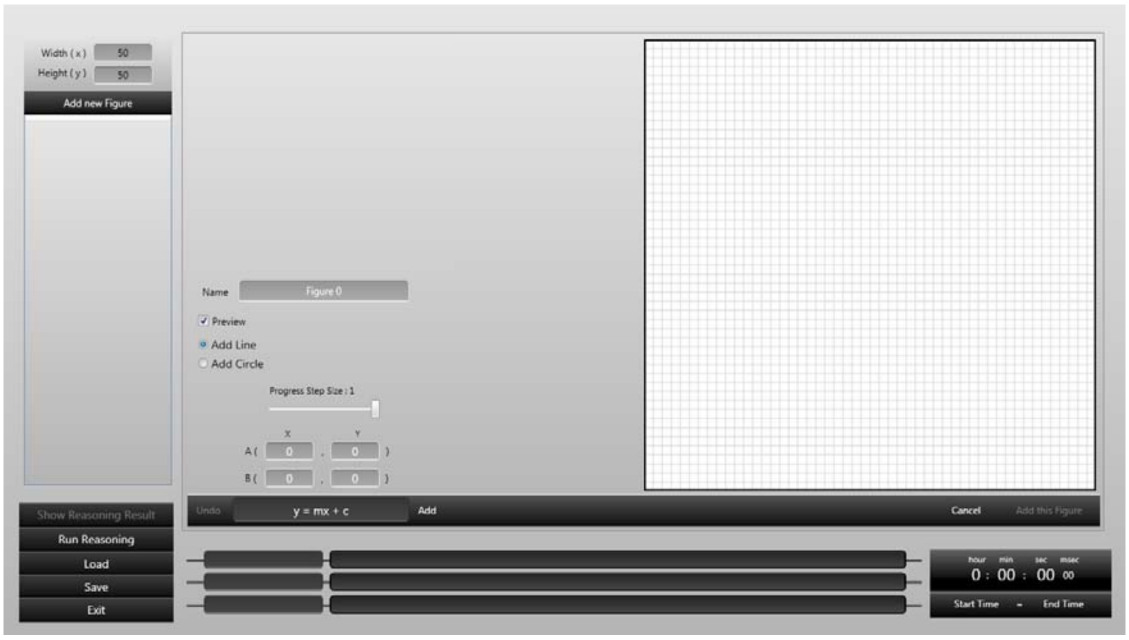

Width (x) 50
Height (y) 50
Name Figure 0
[✓ Preview]
[✓ Add Line]
Add Circle
Progress Step Size : 1
A (0 , 0)
B (0 , 0)
y = mx + c
hour min sec msec
0 : 00 : 00 00

Figure A7.2 Manually defining new shape interface

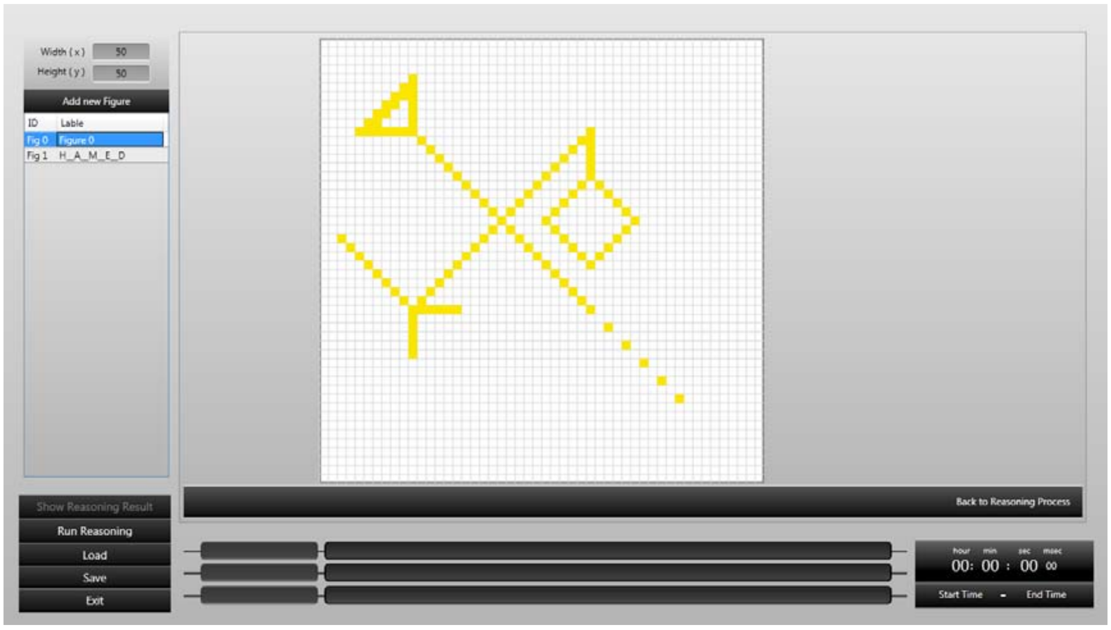
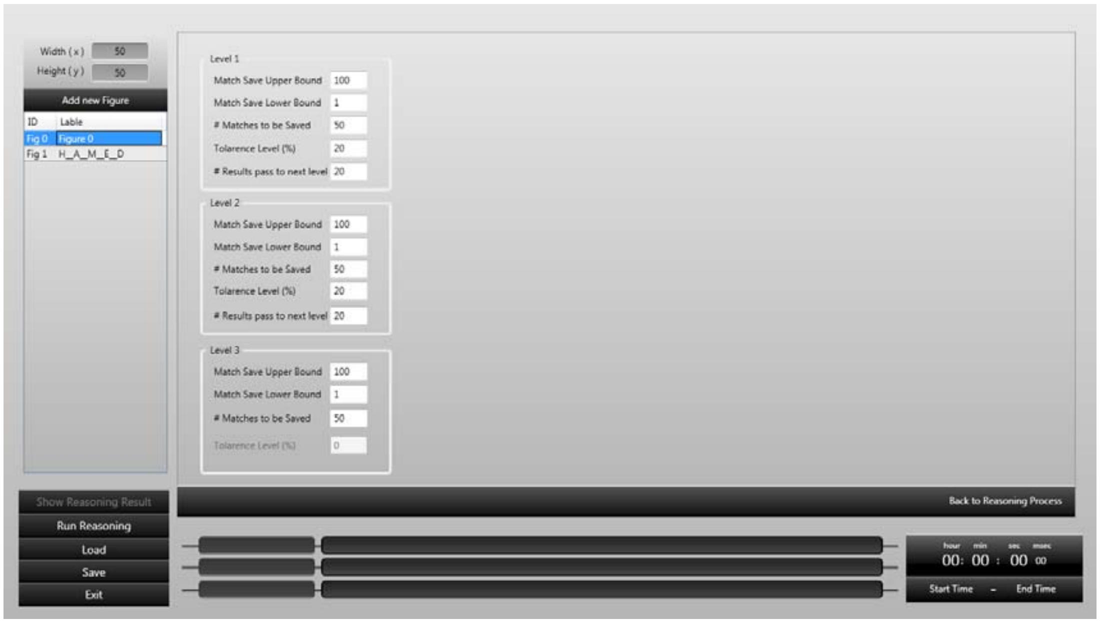


Figure A.3 Review a defined shape



Width (x) 50
Height (y) 50

Level 1

Match Save Upper Bound	100
Match Save Lower Bound	1
# Matches to be Saved	50
Tolarence Level (%)	20
# Results pass to next level	20

Level 2

Match Save Upper Bound	100
Match Save Lower Bound	1
# Matches to be Saved	50
Tolarence Level (%)	20
# Results pass to next level	20

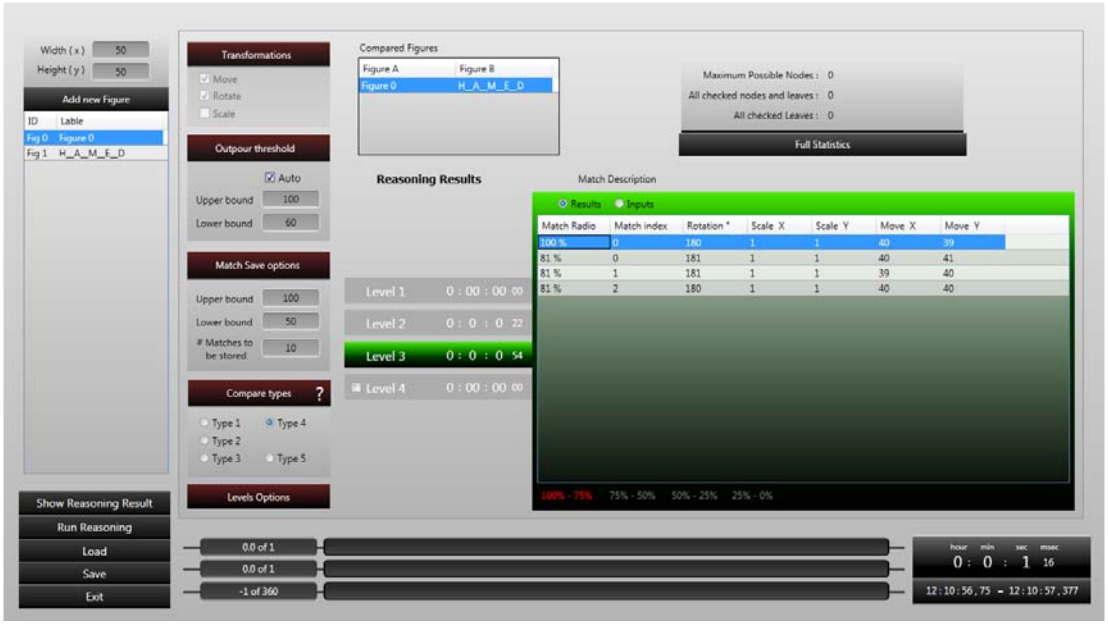
Level 3

Match Save Upper Bound	100
Match Save Lower Bound	1
# Matches to be Saved	50
Tolarence Level (%)	0

Figure A.4 The interface of setting up options for each of the levels separately

Having finished defining figures and setting the desired values for options, clicking of Run Reasoning button runs reasoning process on the set of the defined shapes. This process is programed as multi-thread which keeps the form responding while running reasoning which allows us to review the figures during the reasoning task being run. After the reasoning process is finished, results are ready for view. An example is given in Figure A.5.

Figures are compared in couples, and the Couples are shown in Compared Figures section, choosing each of them we will be able to view the results of the leveling process of each. On Figure A.5, the compare result on level three for the selected couple is shown. Match ratio, Rotation angle and Translation for each of the results are displayed. Double clicking on any of the transformation, the selected transformations will be illustrated by an animation, see Figure A.6.


Width (x) 50
Height (y) 50

ID	Lable
Fig 0	Figure 0
Fig 1	H_A_M_E_D

Transformations: [✓ Move] [✓ Rotate] Scale

Outpour threshold: [✓ Auto]
Upper bound 100
Lower bound 60

Match Save options:
Upper bound 100
Lower bound 50
Matches to be stored 10

Compare types: Type 1, Type 2, Type 3, [✓ Type 4], Type 5

Compared Figures

Figure A	Figure B
Figure 0	H_A_M_E_D

Maximum Possible Nodes : 0
All checked nodes and leaves : 0
All checked Leaves : 0

Reasoning Results

Level 1 0 : 00 : 00 00
Level 2 0 : 0 : 0 22
Level 3 0 : 0 : 0 54
Level 4 0 : 00 : 00 00

Match Description: [✓ Results] Inputs

Match Radio	Match index	Rotation °	Scale X	Scale Y	Move X	Move Y
100 %	0	180	1	1	40	39
81 %	0	181	1	1	40	41
81 %	1	181	1	1	39	40
81 %	2	180	1	1	40	40

0.0 of 1
0.0 of 1
-1 of 360

hour min sec msec
0 : 0 : 1 16
12 : 10 : 56 , 75 - 12 : 10 : 57 , 377

Figure A.5 An example of comparing a couple of shapes

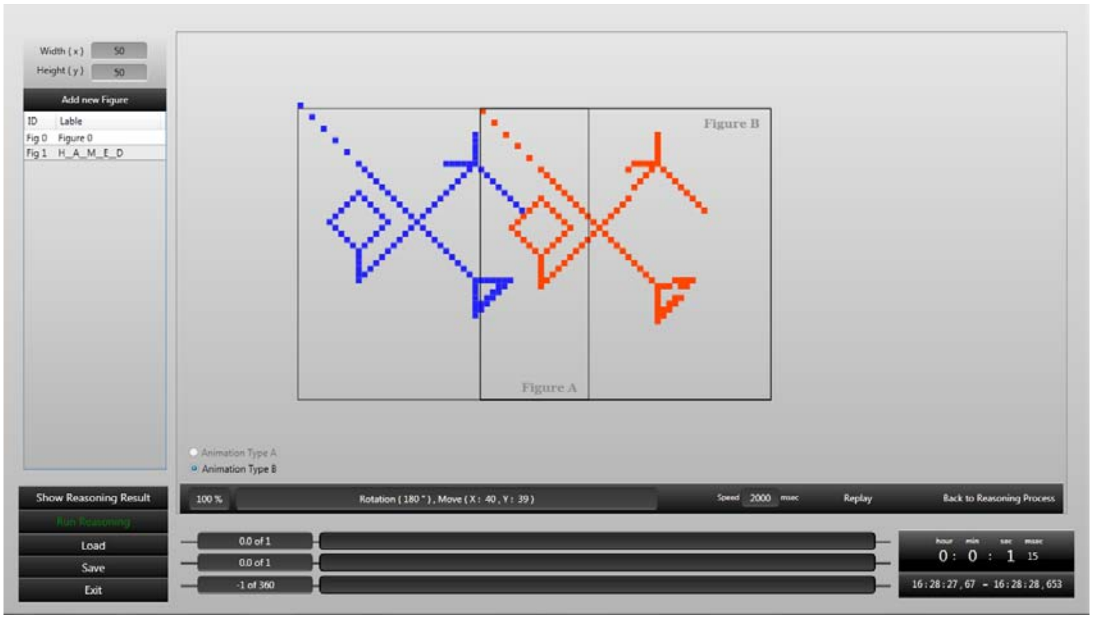
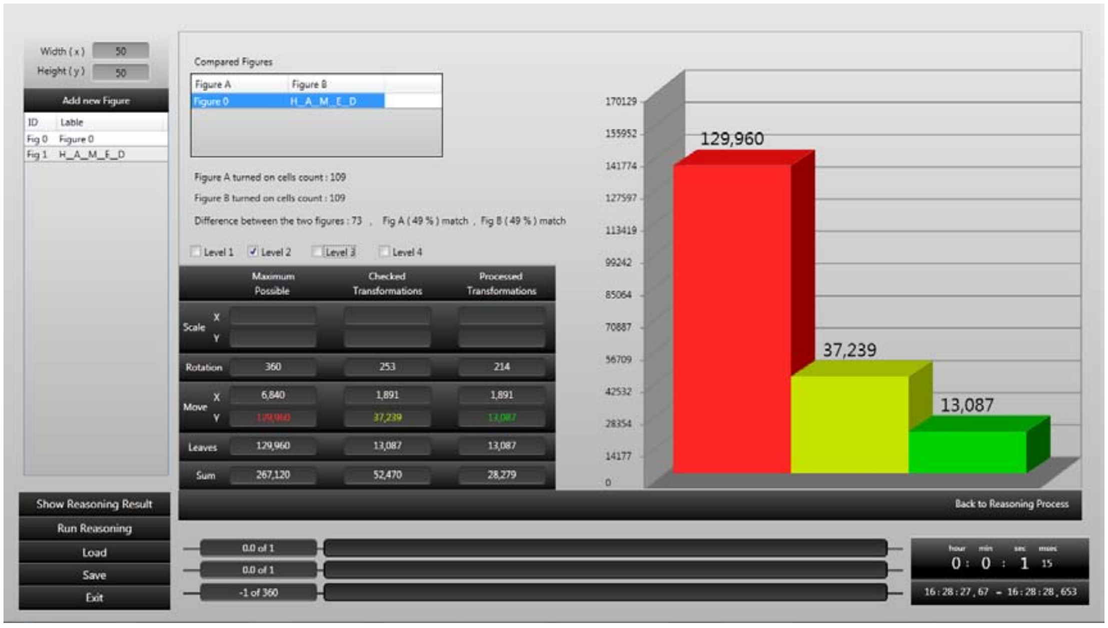


Figure A.6 The selected transformation is illustrated by an animation

The statistics of comparing each of the couples is also available for view by clicking of Full statistics button. Doing so, a widow such as the one shown in Figure A.7 will be displayed with some options of displaying statistics.



Compared Figures

Figure A	Figure B
Figure 0	H_A_M_E_D

Figure A turned on cells count : 109

Figure B turned on cells count : 109

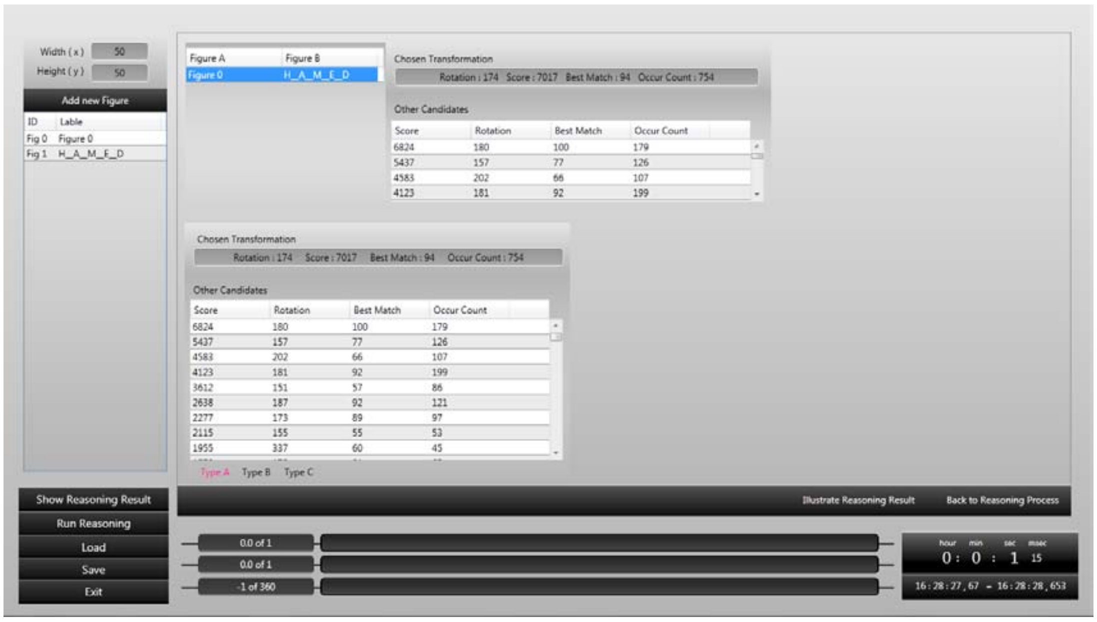
Difference between the two figures : 73 , Fig A (49 %) match , Fig B (49 %) match

☐ Level 1 ☑ Level 2 ☐ Level 3 ☐ Level 4

		Maximum Possible	Checked Transformations	Processed Transformations
Scale	X			
	Y			
Rotation		360	253	214
Move	X	6,840	1,891	1,891
	Y	129,960	37,239	13,087
Leaves		129,960	13,087	13,087
Sum		267,120	52,470	28,279

Figure A.7 View statistics of comparing each of the couples

Clicking on the Show Reasoning Result‘s button, the program uses the best transformations of all pairs to determine a shape which suits best to stand after the last shape in the sequence. The results of this task will be shown on a window such the one in Figure A.8. It informs us which transformation of each pair is chosen and what is the final analysis and also what are other candidates. Clicking on Illustrate Reasoning Results button, the program draws the determined best shape, see Figure A.9, the red shape is the best candidate and blue shapes are the figures in the sequence. Also by clicking on Next Candidate button we could see other candidates as well.



ID	Lable
Fig 0	Figure 0
Fig 1	H_A_M_E_D

Figure A	Figure B
Figure 0	H_A_M_E_D

Chosen Transformation

Rotation : 174 Score : 7017 Best Match : 94 Occur Count : 754

Other Candidates

Score	Rotation	Best Match	Occur Count
6824	180	100	179
5437	157	77	126
4583	202	66	107
4123	181	92	199

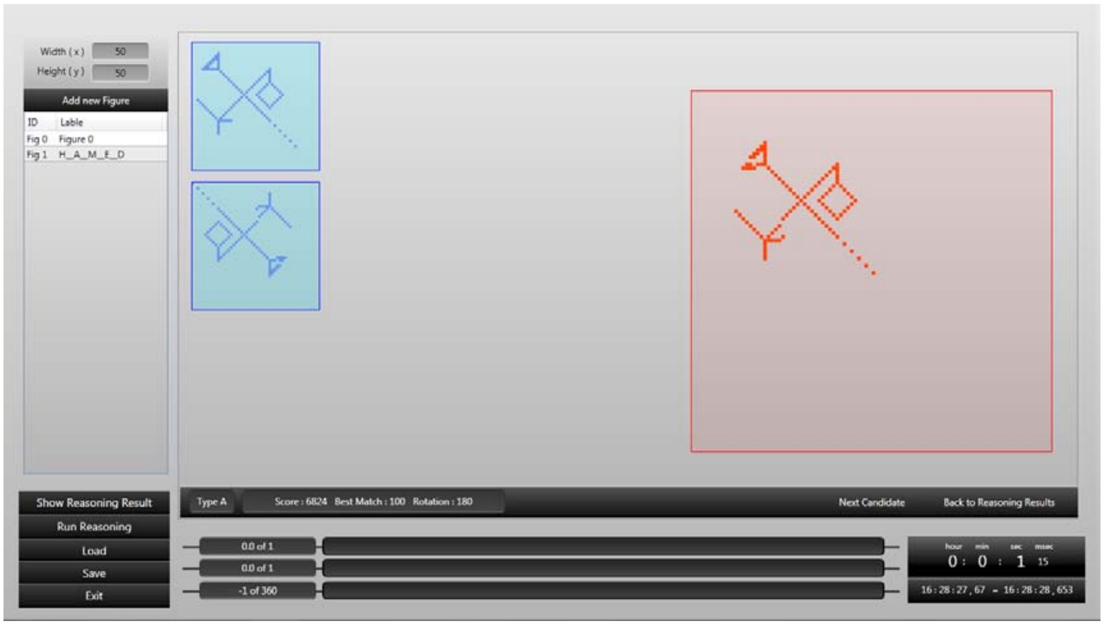
Chosen Transformation

Rotation : 174 Score : 7017 Best Match : 94 Occur Count : 754

Other Candidates

Score	Rotation	Best Match	Occur Count
6824	180	100	179
5437	157	77	126
4583	202	66	107
4123	181	92	199
3612	151	57	86
2638	187	92	121
2277	173	89	97
2115	155	55	53
1955	337	60	45

Figure A.8 Reasoning Results



Width (x) 50
Height (y) 50

ID	Lable
Fig 0	Figure 0
Fig 1	H_A_M_E_D

Score : 6824 Best Match : 100 Rotation : 180

0.0 of 1
0.0 of 1
-1 of 360

0 : 0 : 1 15
16 : 28 : 27 , 67 - 16 : 28 : 28 , 653

Figure A.9 Illustrated reasoning results

TABLE OF FIGURES

Figure 1.1. Distance estimation by using visual information.......................................2
Figure 1.2. To group the points in initial shape, ...4
Figure 1.3. A sample set of Points presenting a shape...13
Figure 2.1. Presenting a sample shape using a matrix 16
Figure 2.2. State space search for the transformation determination. 17
Figure 2.3. A sample of partial match between two shapes is illustrated. 20
Figure 2.4 The manual shape input interface. 23
Figure 4.1 Cropping the Shapes avoids unnecessary search for matches. 25
Figure 3.1 Determine the transformation between these two shapes..........................27
Figure 3.2 A brief visualization of rotation determination procedure. 28
Figure 3.3 Regions and Segments..29
Figure 3.4 Visualization of adding a dimension to 2D segmentation vectors 30
Figure 3.5 Different coordinate systems 32
Figure 3.6 Three sample cases of translation process are illustrated. 34
Figure 3.7 Sample Rotations 36
Figure A.1 Main Window 64
Figure A.2 Manually defining new shape interface 65
Figure A.3 Review a defined shape 66
Figure A.4 The interface of setting up options for each of the levels separately........66
Figure A.5 An example of comparing a couple of shapes 67
Figure A.6 The selected transformation is illustrated by an animation 68
Figure A.7 View statistics of comparing each of the couples.....................................68
Figure A.8 Reasoning Results...69
Figure A.9 Illustrated reasoning results 70