

## Bachelor Thesis

# Comparison of Task Completion using Eye Tracking Recordings

	“	H	E	L	L	O
“	0 ← 1	2	3	4	5	
E	1	1	1	2	3	4
H	2	1	2	2	3	4
L	3	2	2	2	2	3
O	4	3	3	3	3	2
L	5	4	4	3	3	3

### Author

Livio Bereiter

### Supervisor

Felix Wang

### Professor

Prof. Dr.-Ing. Mirko Meboldt

## Bachelor Thesis

# Comparison of Task Completion using Eye Tracking Recordings


## Dates

Start date 19.09.2019

End date 31.01.2020

## Declaration of Originality

I hereby declare that I have written the present thesis independently and have not used other sources and aids than those stated in the bibliography.



---

Livio Bereiter

## Confirmation

This thesis was written at and accepted by **pd|z** Product Development Group Zurich of ETH Zurich.

**Supervisor**

**Professor**

---

---

# Acknowledgement

The elaboration of this thesis wouldn't have been possible without the support and effort of multiple people.

The greatest thanks go to my supervisor Felix Wang, who made it possible for me to deepen my knowledge and experience in programming with Matlab throughout this project. The constant competent inputs and encouragement from his side were of crucial importance and made the collaboration pleasant over the course of the work.

Furthermore, I would like to thank Prof. Dr.-Ing. Mirko Meboldt for his time to discuss my interests for a possible thesis and subsequently introducing me to his research associates. His attitude in terms of teaching and pro-actively interacting with students has inspired me during my studies.

My thanks also go to Luca Vincentini, a reasearch associate at the University Hospital of Zurich, who I met over the course of a previous project. He offered his support and expertise regarding the medical part of this thesis whenever possible. His feedback was critical for the improvement of the skill assessment and enabled the generation of first results with the program.

And finally I would like to thank my family and closest friends who supported and encouraged me throughout the entire project.

Livio Bereiter, January 21, 2020

# Abstract

In many professions, where high reliability and precision is mandatory, the application of new technologies is a recurring event. With these technologies, new techniques come along which must be internalized by the performing person fast and with high quality. Therefore, the demand for efficient and objective training possibilities with skill assessment increases. In the profession of surgery the stated is a daily challenge. Various skill assessment and training concepts are firmly established in medicine, but most of them lack of objectivity, since the performance feedback relies almost fully on the feedback of a single or several procedure experienced surgeons. This system prevents clinic comprehensive skill assessment and does not draw the benefit of novel analysis techniques such as eye tracking technology in order to improve efficiency.

To tackle this problem, a program was developed with MATLAB to analyse surgeons' action sequences generated from eye tracking recordings gathered at the University Hospital of Zurich. The aim of the program is to compare different sequences with suitable algorithms and at the end provide a performance feedback based on the similarity to the optimal sequence of actions to perform a specific surgery. To achieve this, three different algorithms were implemented and tested in terms of objectivity and reliability to calculate a first score deduction by comparing the sequences. In a second step, an algorithm was designed which is able to search for user provided surgery steps, that have been defined to be crucial for the performance feedback. According to this algorithm a second score deduction is carried out, leading to the final surgery score.

The final version of the program allows the automatic generation of skill assessment in terms of surgical action sequences. Once the recordings are gathered and the corresponding sequences are elaborated, the program can give accurate and objective performance feedback in a short amount of time. However, the program's accuracy is highly dependent on the optimal sequence of actions for the surgery to be analysed and the corresponding crucial performance feedback steps. Thus, the greater the intervention expertise of the person providing these the higher the program's reliability. Also, cross-clinical application becomes possible as long as eye tracking is used by the hospitals.

# Contents

<b>Acknowledgement</b>	<b>I</b>
<b>Abstract</b>	<b>II</b>
<b>Contents</b>	<b>III</b>
<b>Nomenklatur</b>	<b>V</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Methods and Materials</b>	<b>4</b>
2.1 Data . . . . .	4
2.1.1 Vessel Catheterization . . . . .	4
2.1.2 Sequence Generation . . . . .	5
2.1.3 Input Files . . . . .	7
2.2 Sequence Comparison Algorithms . . . . .	7
2.2.1 Edit Distance Algorithms . . . . .	7
2.2.2 Common Sub-Sequence Algorithm . . . . .	9
2.2.3 Specific Task-Step Algorithm . . . . .	9
2.3 Surgical Sequence Analysis . . . . .	11
2.3.1 First Score . . . . .	11
2.3.2 Second Score . . . . .	12
2.3.3 Final Score . . . . .	12
2.4 Area of Interest Sequence Analysis . . . . .	13
2.4.1 Similarity . . . . .	13
2.4.2 Appearances . . . . .	14
<b>3 Results</b>	<b>15</b>
3.1 Surgeon Skill Assessment . . . . .	15
3.1.1 Architecture of Program . . . . .	15
3.1.2 Input and Output . . . . .	16
3.1.3 Performance Feedback for Vessel Catheterization . . . . .	17
3.2 AOI Analysis for Usability Studies . . . . .	20

3.2.1	Architecture of Program	20
3.2.2	Input and Output	20
<b>4</b>	<b>Discussion</b>	<b>21</b>
4.1	Surgical Sequences	21
4.2	AOI Sequences	22
<b>5</b>	<b>Conclusion</b>	<b>24</b>
<b>A</b>	<b>Results of AOI Sequence Analysis</b>	<b>26</b>
<b>B</b>	<b>Source Code</b>	<b>38</b>
B.1	Surgical Sequence Analysis Code	42
B.2	AOI Sequence Analysis Code	124
B.3	Functions/Algorithms for Sequence Analysis	155
	<b>List of Figures</b>	<b>174</b>
	<b>Codeverzeichnis</b>	<b>175</b>
	<b>Bibliography</b>	<b>177</b>

# Nomenklatur

## Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
<i>pd z</i>	Product Development Group Zurich
AOI	Area of interest
USZ	University Hospital of Zurich

# 1 Introduction

Medical progress is strongly linked to the constant development and application of novel technologies. It determines for example a surgeon's possibilities to treat patients and ensures an increase in high quality surgical training tools. A current example for the former is the telemonitoring of patients' information, with which treatments at a distance became possible for people carrying pacemakers or internal defibrillators (Saner (2014)). The increase in surgical training tools however, is just as important, since new ways of treatments are presented consecutively and therefore surgeons face the daily challenge of learning complex skills needed to master these. During and after acquiring these new skills, it is essential to provide a method of skill assessment as a component of medical training since it improves effectiveness (Ahmed et al. (2011)). Therefore multiple tools were developed and tested to provide such. Ahmed et al. distinguished between tools which used generic assessment scales to evaluate generic technical abilities of surgeons and procedure-specific scales to breakdown a procedure into tasks (Ahmed et al. (2011)). Similarly Larson et al. proposed and tested an operative performance rating system to evaluate surgeons (Larson et al. (2005)). These rating tools all had at least one commonality: The evaluation was always executed by a set of expert surgeons with or without predefined checklists and without the application of novel technologies as auxiliaries. This resulted in difficulties to provide reliable, objective and cross-clinical skill assessment. Additionally, due to work time regulations the experts' time for supervision of surgical training became increasingly limited and therefore the demand for efficient learning tools increased as well. For these reasons the need for a tool tackling these improvement points arose (Ahmed et al. (2014)).

With the advancement and affordability of eye tracking glasses for medical institutions and the following methods to analyse the gathered data, opportunities to cover these demands became available (Harezlak & Kasprowski (2018)). Eye tracking methods are generally used to record eye movements and find the direction and target of a participant's gaze.



These recordings can then give useful information about a subject's emotions, intentions and most importantly for skill assessment how its knowledge and skills are applied (Harezlak & Kasprowski (2018)). More specifically, two main eye movement elements can be extracted, fixations and saccades. Fixations are defined as an instant when the participant's eyes are almost stable, and saccades are very short and fast eye movements in between fixations.

The aim of the technologies' application in medicine (amongst others) is to gain better understanding of experts' visual activity, to improve educational methods and increase the efficiency for training for less experienced surgeons (Harezlak & Kasprowski (2018)).

One possible way of skill assessment is to compare scan paths of different surgeons with different levels of expertise. In previous research the approaches differ from statistical analysis of the surgical recordings' gaze overlay difference (Khan et al. (2012)) to the comparison of sequences with algorithms, which calculate a distance measure, for example the Levenshtein distance (Kok et al. (2016)). Other work in this direction includes the use of several eye tracking metrics and the comparison of those with linear discriminate analysis and nonlinear neural network analysis (Richstone et al. (2010)).

The quality and significance of such assessments however is strongly depending on the approaches and algorithms used to analyse the data. A Levenshtein distance algorithm for example is constrained to the three possible operations substitution, insertion and deletion, it can execute to count the amount of adjustments necessary to change a letter in a word into another.

In order to enlarge the possibilities in skill assessment using surgical eye tracking recordings, this thesis aims to provide a novel tool to tackle this topic. As a first part of the thesis, a program using MATLAB was developed with which it is possible to analyse the surgeons' action sequences and to give an objective, quantitative performance feedback using only the person of view recordings. The goal is to find amongst different edit-distance algorithms and pattern-finding algorithms the best suitable ones in terms of quantitative, objective, reliable and surgeon specific feedback. Examples for such algorithms are the Levenshtein distance, Damerau-Levenshtein Distance or the Longest Common Subsequence algorithm. Simultaneously the program is constructed dynamically, to ensure the possibility of analysing different data sets and of giving skill assessment for various procedures.

In a second step the variability of the program is verified by using area of interest (AOI) sequences. These were taken from a usability study of a medical device using mobile eye tracking. This ensures the applicability of

the program to multiple scopes where sequence analysis is of interest and approves the dynamic concept of it.

## 2 Methods and Materials

### 2.1 Data

In order to develop the sequence analysis program, a surgical procedure with a manageable number of steps had to be used. Since previous research done by the Product Development Group Zurich (pd|z) focused on minimally invasive cardiovascular surgeries and eye-tracking recordings of these were already gathered, such a surgery was chosen to generate a first set of sequences.

#### 2.1.1 Vessel Catheterization

The catheterization of the femoral artery and femoral vein at the groin of the patient is the first sub-procedure a surgeon does perform as part of a so called MitraClip procedure.

During this minimally invasive cardiovascular intervention, the surgeon intends to place a clip at the mitral valve, which is a heart valve that separates the left atrium from the left ventricle in the human heart. Since the surgery is minimally invasive, the surgeon uses long catheters, entering the patient's body at the groin area, to place the clip at its desired position. To enable the insertion of the long catheters, the surgeon previously needs to catheterize the vessels at the groin area with a slightly adapted version of the so called Seldinger technique (Ivar Seldinger (2008)). The Seldinger technique itself consists of six steps to successfully catheterize a human vessel which are explained in figure 2.1.

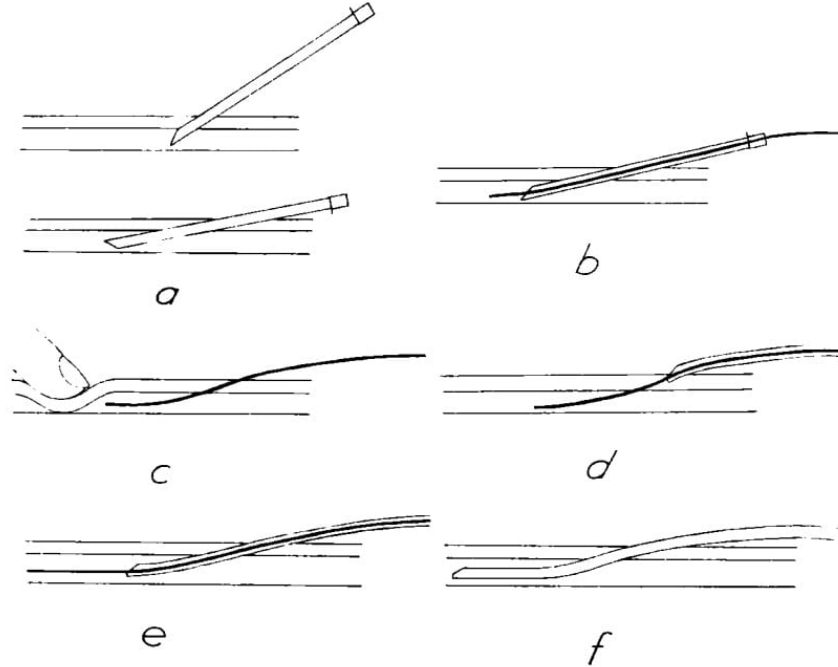


Fig. 2. Diagram of the technique used. a) The artery punctured. The needle pushed upwards. b) The leader inserted. c) The needle withdrawn and the artery compressed, d) The catheter threaded on to the leader. e) The catheter inserted into the artery. f) The leader withdrawn.

Figure 2.1: Intervention steps to successfully catheterize a vessel according to Seldinger technique(Ivar Seldinger (2008)).

### 2.1.2 Sequence Generation

For the catheterization of the femoral artery and vein, the surgeon performs some additional steps which are characteristic for the sub-procedure. In order to capture every step needed for a successful catheterization with all the procedure specific steps, eight novice recordings and nine expert recordings were manually analysed. During the analysis each step was collected in an excel table with the corresponding time for the whole catheterization.

Once all the recordings were analysed each step was labeled with a specific letter or number, which lead to the surgery sequences. The specific labelling and an example for a sequence are visualized in figure 2.2 and 2.3.

Surgery-Steps	Abbreviations	Surgery-Steps	Abbreviations
Left leg (Artery)		Right Leg (Vein)	
Localization of artery (with Us)	a	Attachment of needle to syringe	t
Localization of artery (with fingers)	b	Localization of vein (with Us)	u
Puncture of artery	c	Localization of artery (with fingers)	b
Insertion of guide wire	d	Puncture of vein	v
Verification of access with fluoroscopy	e	Removal of syringe	w
Removal of needle	f	Insertion of guide wire	d
Insertion of catheter with dilator	g	Verification of access with fluoroscopy	e
Removal of guide wire and dilator	h	Removal of needle	f
Flooding of catheter	5	Widening of puncture with scalpel	x
		Insertion of TSP catheter	y
<b>Individual steps at LL:</b>		<b>Individual steps at RL:</b>	
(Removal of guide wire)	4	(Removal of guide wire)	4
(Spreading of cut with tool)	k	(Spreading of cut with tool)	k
(Removal of everything)	l	(Insertion of sewing thread)	z
(Insertion of special, long dilator)	m	(Localization of vein with fluoroscopy)	1
(Attachement of special pipe)	n	(Insertion of small dilator to widen access)	2
(Localization of artery with fluoroscopy)	o	(Insertion of large dilator to widen access)	3
(Catheterization of artrey skipped)	p		
(Switch back to LL and puncture vein there)	q		
(Insertion of additional US probe in LL vein)	r		
(Removal of dilator)	s		

Figure 2.2: Specific labelling of the femoral artery and vein catheterization steps performed previously of the MitraClip intervention.

P08\_3

acdefgs54

tuvwxyefxy

Figure 2.3: Example sequence of a surgery according to beforehand defined labelling; Middle: Left leg sequence; Right: Right leg sequence.

Since there is a characteristic separation, where the surgeon operates first at the left leg and then at the right leg of the patient, the sequences have also been generated separately for each leg in order to increase the performance feedback's accuracy and the level of detail.

Based on expert sequences and in collaboration with the University Hospital of Zurich (USZ), a template sequence as in figure 2.4 for a correct and smooth catheterization was generated accordingly.

T1

acdefgh5

tuvwdefxy

Figure 2.4: Template sequence for a correct and smooth vessel catheterization; Middle: Left leg sequence; Right: Right leg sequence.

### 2.1.3 Input Files

The template sequence as well as the sequences of each surgeon are stored in separate .txt files. For the surgeon files a certain layout must be kept. Each row represents a complete vessel catheterization and the second and third column represent the left leg and right leg separation.

## 2.2 Sequence Comparison Algorithms

To meet the challenge of analysing sequences objectively and reliable three different algorithms were chosen and implemented in MATLAB R2019a. The whole code with the different algorithms and functions can be found in the appendix [B](#).

### 2.2.1 Edit Distance Algorithms

Edit distance algorithms are a common approach to quantify how similar two words, in programming language strings, or even whole texts are to each other ([Ristad & Yianilos \(1998\)](#), [Su et al. \(2008\)](#)).

There are two algorithms that were of interest for this project in terms of feasibility due to time limitations. The Levenshtein distance, which is the most common metric in order to compare two strings and the Damerau-Levenshtein distance.

#### Levenshtein Distance

The Levenshtein distance algorithm calculates the minimum number of operations needed to change one string into another desired string. It is able to perform one out of three possible operations, which are insertion, replacement or deletion, in order to change a single character in a string into another ([Levenshtein \(1966\)](#)).

An example on how the algorithm works is presented in figure [2.5](#). Each number to the left, upper left and top of the cell, where the algorithm currently compares two characters, represents the cost for the respective operation, deletion (left), replacement (upper left) and insertion (top).

The first row and column of the matrix are filled with the minimum number of operations needed to generate each string out of nothing ("). As a next step the algorithm starts to compare the first character of "ephrem", which is "e", with the first character of "benyam", which is "b". In the respective cell it

then stores the minimum operation-cost to change "b" to "e", which is 0 (cost for replacement), and adds 1 since the characters are not the same, resulting in a total of 1.

This is done correspondingly for each following character of "benyam". Once the algorithm reaches the last character of "benyam", it simply jumps down a row in order to compare the next character of "ephrem", which is "p", with each character of "benyam" again.

By doing so the algorithm provides as a final output on the bottom right of the matrix the overall minimum number of operations needed, the so called edit distance, to change "benyam" to "ephrem".

Replace	Insert							
Delete	Minimal cost (+1)							
		e	b	e	n	y	a	m
e	0	1	2	3	4	5	6	
e	1	1	1	2	3	4	5	
p	2	2	2	2	3	4	5	
h	3	3	3	3	3	4	5	
r	4	4	4	4	4	4	5	
e	5	5	4	5	5	5	5	
m	6	6	5	5	6	6	5	

Figure 2.5: Dynamic matrix (light grey) generated by Levenshtein distance algorithm, where on the bottom right the minimum number of operations to change "benyam" to "ephrem", is presented; Arrows: indicate the optimal reversed path, where each operation is a replacement (diagonal direction).

Applied on the surgery sequences, the algorithm compares the sequences with the respective template and enables a comparison on how similar the surgeon's sequence was to a correct surgery. More specifically, the smaller the minimal number of operations of a surgeon's sequence is, the closer the surgeon is to a correct surgery.

### Damerau-Levenshtein Distance

The Damerau-Levenshtein distance algorithm can perform one additional operation in comparison to the Levenshtein distance algorithm, which is the transposition of two neighbouring characters. Nevertheless, this change has a

great impact on the complexity of the algorithm.

The key component of the algorithm is a variable, which is an array with as many columns as there are characters in the ASCII table. While the algorithm walks through each row of the matrix, similar to the one in figure 2.5, it assigns the current matrix row number of the character it is comparing, to the specific column in the array variable, represented by the character's ASCII number. To make it more clear, if the algorithm would currently be comparing the character "e" of "ephrem" in the matrix row six, it would assign this row number to the column 101 in the array variable, since this is the ASCII number for "e". Because the algorithm only needs to know the last time a character appeared in the desired string, it simply overwrites the beforehand stored matrix row number in the array variable's column 101, which would be two for the first appearance of the character "e".

This allows the algorithm to keep track of which character of "ephrem" appeared the last time in which row of the matrix. Thus, while walking through each row, it can compare the costs for the beforehand explained operations with the cost for a transposition of two neighbouring characters and choose the overall minimum.

### 2.2.2 Common Sub-Sequence Algorithm

The Common Sub-Sequence Algorithm is an open source function, which can be used to find either the longest common subsequence between two strings or a common subsequence with a predefined length(Cumin (2020)).

This can be used to search for characteristic task patterns within the sequences and subsequently make more specific statements about what the participant did wrong in which part of the procedure and how to improve that.

### 2.2.3 Specific Task-Step Algorithm

Depending on the use case or the procedure, the criteria for an accurate performance feedback should be customizable. Therefore, the specific task-step algorithm allows the user to provide an expert's definition of crucial performance feedback steps or subsequences, for which the algorithm searches in each sequence.

More specifically, the expert can provide the steps/subsequences and a short explanation for each of them in form of a .txt file, for which an example is shown in figure 2.6.



Surgery-Steps	Abbreviations
Localization of artery without US	b
Puncture of artery more than once	c
Verification with fluoroscopy forgotten	e
Removal of needle before fluoroscopy verification	fe
Removal of dilator, then guide wire instead of both together	s4

Figure 2.6: Example for an expert's definition of the crucial task steps/subsequences; US: Ultrasound.

If the user of the algorithm wishes to analyse surgical sequences, it asks for an input how to treat each one of the steps/subsequences. There are three different ways how the algorithm can treat the steps/subsequences. It can either distinguish between a step/subsequence is missing in the sequence, it mustn't appear in the sequence or it appears too many times in the sequence. Furthermore, the user can provide a weighting factor for each step/subsequence, which allows to weight each of them according to their impact on the outcome of the surgery. Or in the case that a step/subsequence mustn't appear too many times, the user can also assign the number of its appearances minus the optimal number of appearances as the weighting factor. These weighting factors are then used to carry out deductions of an initial score and lead to a surgery score for each sequence. An example how these user inputs could look like is given in figure 2.7.

If the user wishes to analyse other sequences than surgical sequences, the algorithm simply counts the appearances of any provided steps or subsequences.

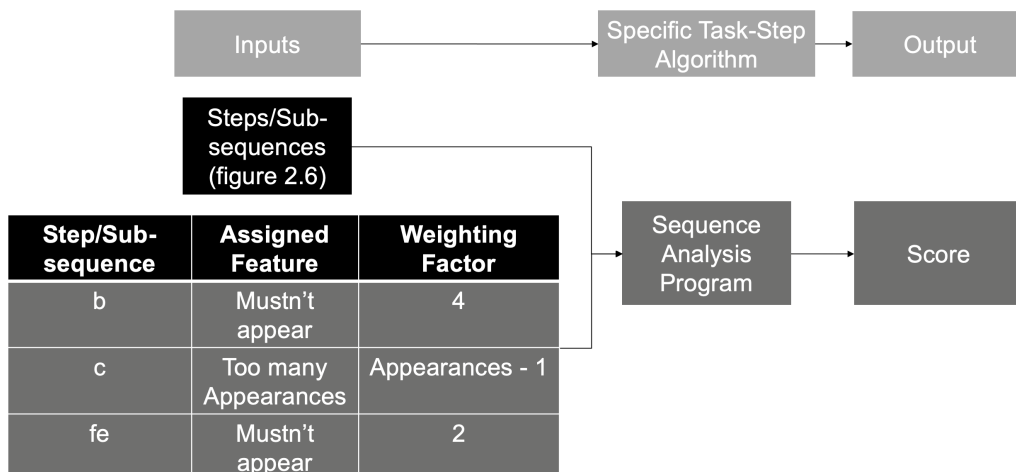


Figure 2.7: Example for user inputs in order to analyse surgical sequences with specific task-step algorithm.

## 2.3 Surgical Sequence Analysis

Since the main purpose of the program is to give surgical performance feedback, an objective and surgeon specific way of giving this skill assessment is required. Therefore, an automatic score calculation for each surgery performed was defined. The following subsections will explain how this was done and which aspects of this calculation are adjustable by the user.

### 2.3.1 First Score

The first score is calculated using the edit distances received by the Levenshtein distance algorithm or Damerau-Levenshtein distance algorithm. As explained in subsection [2.2.1](#) the Damerau-Levenshtein distance algorithm differs from the Levenshtein distance algorithm in terms of possible applicable operations, when comparing the surgeons' sequences with the template sequence. For reasons which will be explained in chapter [4](#), the final version of the program calculates the first score using the edit distances from the Levenshtein distance algorithm.

Each edit distance is compared with four intervals and a respective deduction from an initial score of 100 points is automatically carried out. The four intervals are the following: if the surgeon's surgery sequence got an edit distance of 0, which means the sequence was identical to the template sequence, 0 points are deducted, if the edit distance is 1 or 2, 10 points are deducted, if the edit distance is 3, 4 or 5, 30 points are deducted and in the last case, if the edit distance is 6 or larger, 40 points are deducted.

There are two reasons for this definition of the intervals. Firstly, the impact of the deduction according to the sequence's edit distance is intended to be smaller than the impact of the second deduction carried out by the specific task-step algorithm introduced in subsection [2.2.3](#). Secondly, the first score represents a rough comparison between the surgeons' sequences and the template sequence with the Levenshtein distance algorithm functioning as an objective tool. Therefore, the intervals and deductions in the first score calculation were defined such that the resulting first score can be at least 60 points.

The intervals and deductions are easily adjustable in the code to achieve an optimal score calculation. To make sure the user does so correctly, a description on how to make these changes can be found at the beginning of the code. This is important, since depending on the surgery which is being analysed and the greater expertise an expert surgeon will bring along in defining this first deduction properly, it must be adjustable.

### 2.3.2 Second Score

The second score is calculated using the specific task-step algorithm introduced in subsection [2.2.3](#). Using the user provided crucial performance feedback steps from the corresponding .txt file, the algorithm searches for these steps/-subsequences and carries out a second deduction from the first score according to the inputs of the user. The minimum deduction the user can multiply with the provided weighting factors is 4 points. The reason for this definition of the minimum deduction is, that the accuracy and distinctiveness of the score increases by using a relatively small number in comparison with the deductions carried out in the first score calculation.

As in the first score calculation, the minimum deduction is as well easily changeable in the code if a different procedure has to be analysed or another reason for a change exists. A description is again given at the beginning of the code.

### 2.3.3 Final Score

The final score is composed of the first and second score calculation. After the deductions made in the two beforehand described subsections, the final score is displayed to the user in the so called "Command Window" for every performed surgery. Additionally to the final score, the surgery specific deduction reasons that led to the presented final score, are displayed as well for every surgery. More specifically, the deduction carried out in the first score calculation and the crucial task steps/subsequence-descriptions that led to a deduction in the second score calculation, are presented. These descriptions are provided by the user besides the respective crucial steps/subsequences as can be reviewed in figure [2.6](#).

This is done in order to allow the user to receive more specific feedback on what the final score caused and therefore on where the surgeon needs to improve his skills.

## 2.4 Area of Interest Sequence Analysis

Beside the analysis of surgical sequences, the analysis of AOI sequences from a usability study of a product is another scope of application for the program. These sequences are generated when eye tracking recordings are further investigated and a so-called gaze mapping is carried out, using the eye tracking software BeGaze 3.6. As a first step of the gaze mapping, the AOI's are manually assigned to specific sectors in the participants workspace. In a second step, each time the participant's gaze point enters a previously defined AOI, the user of the software stores the AOI's name in a .txt file. This is done for the whole recording, leading to a list of AOIs. This list is then in a third step transformed with MATLAB into a sequence of characters, each character representing a specific AOI, as shown in figure 2.8.

The complete sequences can then be further analysed with the sequence analysis program and conclusions such as a participant's learning progress in terms of the product handling can be drawn.

To test the programs applicability in this scope, the data from Celine Gianduzzo's bachelor thesis was used and analysed according to the hypotheses put forward by her. Thus one can find more detailed information about the conducted study there (Gianduzzo (2020)).

P7      CCBBBCCAAAAAACCCCBCCCAAAACCCCBCAAAAAAABBAAAAAACCCBBBAAAAAACCC

Figure 2.8: Example for a complete AOI sequence from a usability study; A: Building area; B: Building blocks; C: Computer (Gianduzzo (2020)).

There are two different areas the program can analyse.

### 2.4.1 Similarity

The first one is about similarity between the participants sequences and a template sequence chosen by the user and thereby assigned as the template sequence. The templates differ from an expert's final trial sequence to any trial sequence of an expert or participant of interest, which the user wishes to compare with the trial sequences of the other participants. The sequences need to be provided as well as mentioned in section 2.1.3 in form of .txt files for all participants. The program uses the in subsection 2.2.1 explained Levenshtein distance algorithm to compare the participant's sequences with the template sequence. The lower the edit distance for each sequence is, the closer the sequence is to the chosen template.

This allows the user to use the Levenshtein distance as another metric in order

to approve and visualize the learning progress over different trials during a usability study of a product.

### 2.4.2 Appearances

For the second area, the specific task-step algorithm introduced in subsection 2.2.3 is used in a slightly reduced way. Instead of the crucial performance feedback steps the user can provide AOIs or AOI-patterns. Then, again with inputs asked by the program, the user can define for each AOI/AOI-pattern if the program should find out how many times the abbreviation appears in the sequence or if the abbreviation appears at all in the sequence. The AOIs/AOI-patterns can be provided in form of a .txt file as shown in figure 2.9.

This allows the person conducting the usability study to quickly gain information about the appearance of a crucial or characteristic eye-gaze pattern for the product, or make conclusions about the importance of specific AOIs in terms of visual attention distributed to them during the study.

Type:	Abbreviations:
Pattern of AOI's:	CBA
AOI's:	A
AOI's:	B
AOI's:	C
AOI's:	W

Figure 2.9: Example for user provided AOIs or AOI-patterns to further analyse the sequences (Gianduzzo (2020)).

## 3 Results

In order to present the results of the project the section is separated in the two areas of application of the program, the surgeon skill assessment and the AOI analysis for usability studies.

### 3.1 Surgeon Skill Assessment

#### 3.1.1 Architecture of Program

The architecture of the program is built as visualized in figure [3.1](#). The Data is read and stored in suitable variables for further work with it. In a second step, as explained in section [2.3](#) the first score is calculated with the Levenshtein distance algorithm leading to the first output. The second score is then calculated with the user provided crucial performance feedback steps leading to the second output. The first and second score are then combined to a final score for which the results for the catheterization-procedure are presented in figure [3.5](#) and [3.6](#).

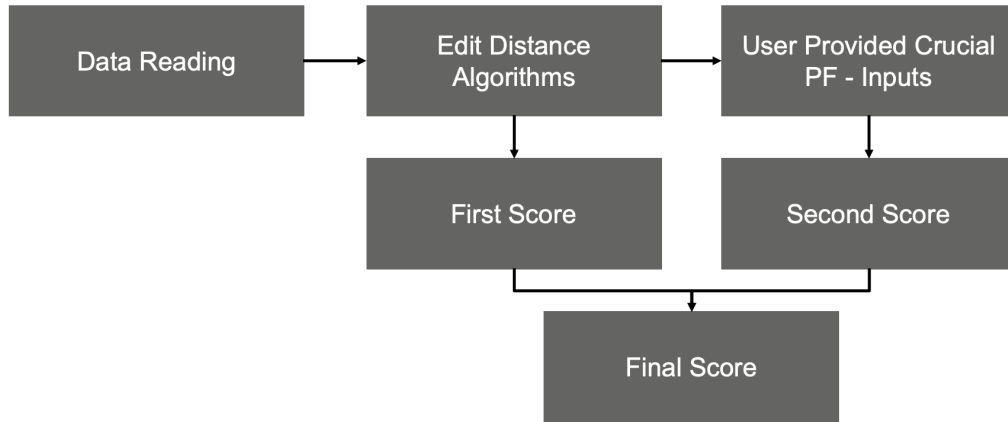


Figure 3.1: Architecture of the surgeon skill assessment program; PF: Performance feedback.

### 3.1.2 Input and Output

As mentioned earlier, all the data and the crucial performance feedback steps are provided in form of .txt files. The input output flow of the program is therefore displayed in figure 3.2 to highlight the dependency of the skill assessment, representing the program's output, of the .txt files. Additionally, it also underlines the variability of the program for different surgical skill assessment. Furthermore, the layout for the final score presentation in the "Command Window" is shown on the right side of figure 3.2, using two chosen vessel catheterizations from Novice1 for presentation purposes.

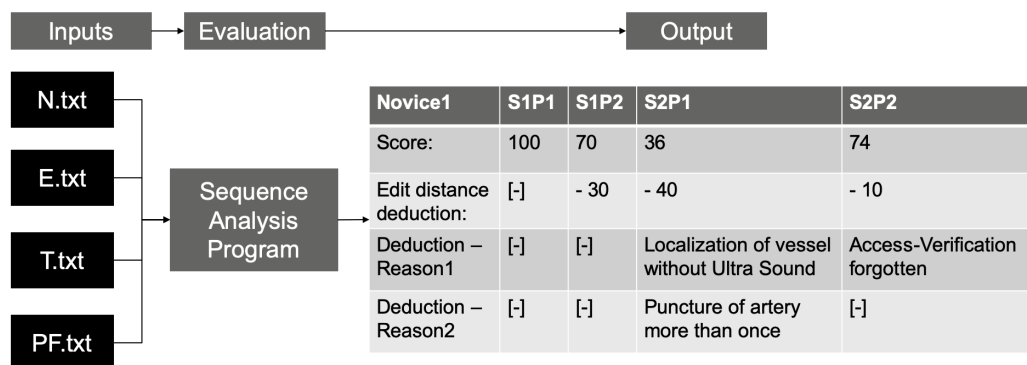


Figure 3.2: Visualization of the .txt file dependency of the program's performance feedback; N: Novice; E: Expert; T: Template; PF: Crucial performance feedback steps; S1P1: Part 1 of surgery 1; S1P2: Part 2 of surgery 1; S2P1: Part 1 of surgery 2; S2P2: Part 2 of surgery 2.

### 3.1.3 Performance Feedback for Vessel Catheterization

For the second score calculation a first set of crucial performance feedback steps/subsequences and respective weighting factors were defined in collaboration with a research associate of the Zurich heart centre at the USZ. Using the data shown in figure 3.3 and 3.4 the program's performance feedback for the surgeons is shown as mentioned earlier in figure 3.5 and 3.6.

Surgery-Steps	Abbreviations	Weighting factor
Localization of artery without US	b	4
Puncture of artery more than once	c	number of appearances
Verification with fluoroscopy forgotten	e	4
Removal of needle before fluoroscopy verification	fe	2
Removal of dilator, then guide wire instead of both together	s4	1

Figure 3.3: Crucial performance feedback steps/subsequences and weighting factors for the left leg catheterization.

Surgery-Steps	Abbreviations	Weighting factor
Verification of access with fluoroscopy forgotten	e	6
Puncture of vein more than once	v	number of appearances
Widening of puncture with scalpel forgotten	x	3
Removal of needle before fluoroscopy verification	fe	2
Localization of artery without US	b	4

Figure 3.4: Crucial performance feedback steps/subsequences and weighting factors for the right leg catheterization.



	<b>S1P1</b>	<b>S1P2</b>	<b>S2P1</b>	<b>S2P2</b>	<b>S3P1</b>
<b>ScoresN1:</b>	90	90	100	100	100
<b>LevDistDeduction:</b>	-10	-10	0	0	0
<b>ScoresN2:</b>	[]	44	[]	[]	[]
<b>LevDistDeduction:</b>	[]	-40	[]	[]	[]
<b>DeductionReason1:</b>	[]	"Puncture of vein more than once"	[]	[]	[]
<b>ScoresN3:</b>	58	90	90	70	54
<b>LevDistDeduction:</b>	-30	-10	-10	-30	-30
<b>DeductionReason1:</b>	"Removal of needle before fluoroscopy verification"	[]	[]	[]	"Verification with fluoroscopy forgotten"
<b>DeductionReason2:</b>	"Removal of dilator, then guide wire instead of both together"	[]	[]	[]	[]
<b>ScoresE1:</b>	44	74	44	90	74
<b>LevDistDeduction:</b>	-40	-10	-40	-10	-10
<b>DeductionReason1:</b>	"Localization of artery without ultra sound"	"Localization of artery without ultra sound"	"Localization of artery without ultra sound"	"Verification with fluoroscopy forgotten"	
<b>ScoresE2:</b>	100	54	50	70	36
<b>LevDistDeduction:</b>	0	-30	-30	-30	-40
<b>DeductionReason1:</b>	[]	"Localization of artery without ultra sound"	"Localization of artery without ultra sound"	[]	"Localization of artery without ultra sound"
<b>DeductionReason2:</b>	[]	[]	"Puncture of artery more than once"	[]	"Puncture of artery more than once"

Figure 3.5: First half of the catheterization scores and deduction reasons for Novice1, Novice2, Novice3, Expert1, Expert2.

	S3P2	S4P1	S4P2	S5P1	S5P2
ScoresN1:	100	0	0	0	0
LevDistDeduction:	0	0	0	0	0
ScoresN2:	0	0	0	0	0
LevDistDeduction:	0	0	0	0	0
DeductionReason1:	0	0	0	0	0
ScoresN3:	70	54	100	0	0
LevDistDeduction:	-30	-30	0	0	0
DeductionReason1:	0	"Verification with fluoroscopy forgotten"	0	0	0
DeductionReason2:	0	0	0	0	0
ScoresE1:	70	74	90	0	0
LevDistDeduction:	-30	-10	-10	0	0
DeductionReason1:	0	"Verification with fluoroscopy forgotten"	0	0	0
ScoresE2:	54	100	44	100	90
LevDistDeduction:	-30	0	-40	0	-10
DeductionReason1:	"Localization of artery without ultra sound"	0	"Localization of artery without ultra sound"	0	0
DeductionReason2:	0	0	0	0	0

Figure 3.6: Second half of the catheterization scores and deduction reasons for Novice1, Novice2, Novice3, Expert1, Expert2.

## 3.2 AOI Analysis for Usability Studies

### 3.2.1 Architecture of Program

For the AOI sequence analysis the program architecture varies slightly. Instead of the first and second score, the program delivers from the Levenshtein distance algorithm the edit distances between the participants' sequences and assigned template. And instead of the crucial performance feedback steps the specific task-step algorithm uses the user provided AOIs and AOI-patterns to deliver the number of their appearances. For a better understanding this is as well visualized in figure [3.7](#)

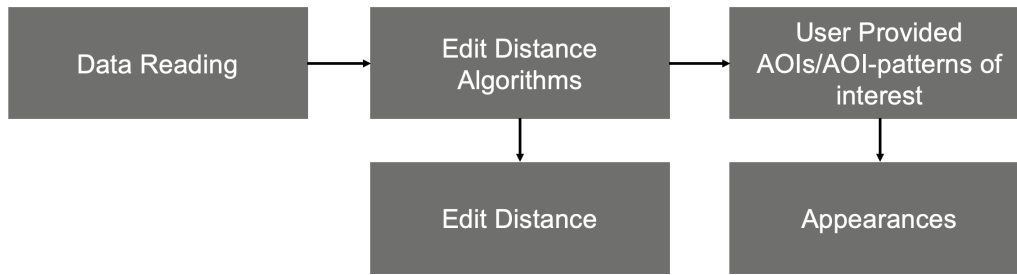


Figure 3.7: Program architecture of the AOI sequence analysis.

### 3.2.2 Input and Output

The input output flow is the same as explained in subsection [3.1.2](#) except that the content of the .txt files is different and respectively the output looks slightly different. The complete results of the AOI sequence analysis for the usability study conducted by Celine Gianduzzo are presented in the appendix [A](#). In figure [A.2](#) for example, the number of appearances of three specific AOI-patterns are shown. Furthermore, the Levenshtein distances (GLD) from two consecutive trial sequences are presented in figure [A.8](#).

## 4 Discussion

### 4.1 Surgical Sequences

The results as presented in section 3 for the surgeons' vessel catheterization show that the program is able to automatically take the provided data and user inputs and transfer them (as defined by the user) into a quantitative and objective way of skill assessment. Especially the reasons provided to each deduction of the final score, make it possible for the user to receive accurate and reliable performance feedback according to the provided step descriptions.

Nevertheless, the experts' scores are atypically low for experts. There are two reasons for this. Firstly, the designation as expert was given for a certain number of performed MitraClip surgeries, of which the vessel catheterization is a sub-procedure. Secondly, the crucial performance feedback steps and respective weighting factors are defined according to observations and thereby represent an optimal/most effective way to perform the catheterization. This is particularly evident for the crucial performance feedback step "b", which stands for "Localization of artery without ultrasound", which is mostly replaced by the expert surgeons with another technique where they localize the artery with their fingers instead. If performed correctly the different technique is not wrong but, according to the observations of the research associate at the USZ, less efficient than the usage of ultrasound. Therefore, the decision was made to add this step to the first set of crucial performance feedback steps, which leads to an additional deduction for most of the surgeries performed by the experts. Furthermore, patient specific aspects, which could make the catheterization harder, such as obesity or naturally small vessels couldn't be taken into account.

Another observation is that the score calculation itself, with the first deduction made in the first score calculation and then another deduction made according to the crucial steps, leads to a quite strict skill assessment. Since the main goal of the project was to develop a program that can automatically analyse and compare sequences in an objective way, further tests in which

different approaches to calculate the final score for the performance feedback, were not feasible in the short time available. Therefore, the whole final score calculation, with the defined edit distance intervals for the first deduction and the multiplicable minimum deduction for the second deductions, are not yet optimally set for an overall reliable performance feedback.

The decision to use the edit distance from the Levenshtein distance algorithm as the first objective comparison tool for the sequences might be another limitation of the program. In comparison with the Damerau-Levenshtein distance algorithm it emerged, that for the sequence comparison, the difference by adding an additional feature wasn't as impactful as expected. This became clear when the edit distance, computed by the Damerau-Levenshtein distance algorithm, was almost always one edit distance less than the edit distance for the same sequence calculated by the Levenshtein distance algorithm. Thus, between these two edit distance algorithms the Levenshtein distance algorithm provides sufficiently accurate results for this application.

However, in terms of the most accurate objective comparison tool, a more reliable algorithm might exist. The applicable operations of the Levenshtein distance algorithm itself are not as limiting as the fact that by analysing surgical sequences, patient specific aspects can't be considered by it. An example for this is the resulting sequence for an obese patient. Such sequences are much longer, since the surgeon has a hard time to find and puncture the vessel below the thick fat tissue. Hence, this does not necessarily represent a bad performance by the surgeon, but a complex surgery and leads to a large edit distance since the algorithm has to add multiple characters in comparison to the template sequence.

## 4.2 AOI Sequences

The successful application of the program to analyse AOI sequences instead of surgical sequences shows that as long as the user supplies his data and specific AOIs/AOI-patterns in form of .txt files with the requested layout, the program can be used for different scopes where sequence analysis is of interest.

However, there are some limitations considering the structure of the provided data. One example for that is the limited possibility to separate the trials from a usability experiment into maximum two parts. A potential solution for that could be the development of a function, which designs the structure of the variables where the provided data gets assigned to, according to inputs from the user before the actual analysis starts.

Furthermore, the output of the program can become quite confusing with an increasing number of participants. Thus, a more intuitive way to visualize the output, like the automatic generation of a suitable plot, could be helpful to prevent the user from losing track.

## 5 Conclusion

Because of its relatively low complexity and the outlined points at the end of section [4.1](#), the Levenshtein distance algorithm provides a quite robust and accurate tool for the task of a rough and objective sequence comparison.

Nevertheless, the Damerau-Levenshtein algorithm is still attached in the code and should be further tested with other sequences and cases, to approve the observations made throughout this project.

The same applies to the Longest Common Subsequence algorithm. Within the scope of this project, the algorithm isn't directly involved in the performance feedback but as well attached to the code. Especially for the analysis of more complex surgeries and thus longer sequences, this algorithm could provide additional possibilities. For example, a surgery specific pattern analysis to provide the user with more detailed feedback about specific areas a surgeon should further improve, could be enhanced.

One factor all these algorithms can't consider, is the variability of the interventional complexity due to patient specific aspects. Therefore, to ensure a more reliable performance feedback for surgeons with this program, the future patients and hence the in the future gathered data, must be preselected to reduce the probability of imprecise feedback.

As mentioned earlier in section [4.1](#) the score calculation is not yet at a state where an overall reliable skill assessment is achieved. Thus, further work should be invested in the elaboration of the optimal edit distance interval definition for the first score deduction and the definition of the multiplicable minimum deduction for the second score deduction. Additionally, the crucial performance feedback step file for the vessel catheterization ought to be elaborated in collaboration with expert surgeons from the USZ. Moreover, it would be beneficiary to attach the respective weighting factors to the crucial steps/subsequences in order to reduce the amount of inputs the user has to give. Once these adjustments are carried out, a larger data set and hence more surgery sequences should be gathered to test and approve the program's robustness and reliability.

Nevertheless, the structure of the program's performance feedback with the presentation of the final score and the reasons for the deductions provides any user with a quantitative, objective and surgeon specific skill assessment.

Also, with the successful analysis of AOI sequences, the program's dynamic concept was approved and in terms of sequence analysis, its rather use case independency was confirmed, as long as the data is provided in .txt files with the requested layout.

To sum it up, a solid foundation for an automatic computer tool was developed, which allows (besides others) the analysis of surgical action sequences in an objective manner and provides the user with a quantitative and task specific performance feedback.



## A Results of AOI Sequence Analysis

The figures [A.1](#) - [A.11](#) represent the results of the AOI sequence analysis conducted in the scope of Celine Gianduzzo's bachelor thesis([Gianduzzo \(2020\)](#)).

The AOI pattern designations 'Learning', 'Memorized 2' and 'Memorized 3' represent CBA, BABA and BABABA respectively. Which character for which AOI stands is explained in figure [2.8](#). Additionally, GLD is short for General Levenshtein Distance.

### 1.3 Gaze pattern occurrence

By counting gaze patterns ('Learning', 'Memorized 2', 'Memorized 3' as defined in ??) in the AOI-visit sequences, the number of pattern occurrences was calculated for each trial and every novice (regular participant). In addition, the same approach was applied to the two experts of the related groups.

The two groups, 'Simple' and 'Complex', were evaluated separately. The overall results are shown in Figure 1.7 for group 'Simple' and Figure 1.8 for group 'Complex'. The charts represent the gaze pattern counts averaged over all 14 novices and 8 trials of respective study group. Moreover, they show the pattern occurrences for the experts of corresponding group type. The solid lines represent the novices' data (N-Learning, N-Memorized 2, N-Memorized 3) and the dotted ones represent the experts (E-Learning, E-Memorized 2, E-Memorized 3). The x-axis gives the number of trials and the y-axis the number of pattern occurrences calculated.

#### 1.3.1 Group 'Simple'

The 'Learning' pattern for the novices (N-Learning) decreases starting at trial 4 (from  $9.4 \pm 3.3$  occurrences at trial 4 to  $4.7 \pm 4.2$  occurrences reaching trial 8). The N-Memorized 2 and N-Memorized 3 both increase steadily from the beginning over all following trials. The N-Learning and N-Memorized 2 curves intersect at trial 6. The N-Learning and N-Memorized 3 curves intersect in between trial 7 and 8 (see Figure 1.7).

The expert of group 'Simple' shows a different gaze behavior. The E-Learning pattern occurs only in the sequence of the first trial and remains on the level 0 for the remaining ones. By contrast, E-Memorized 2 and E-Memorized 3 are constantly present patterns over all 8 trials and occur on average  $20.4 \pm 1.3$  and  $17.9 \pm 2.2$  times per sequence (averaged over all trials) (see Figure 1.7).

Figure A.1: Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

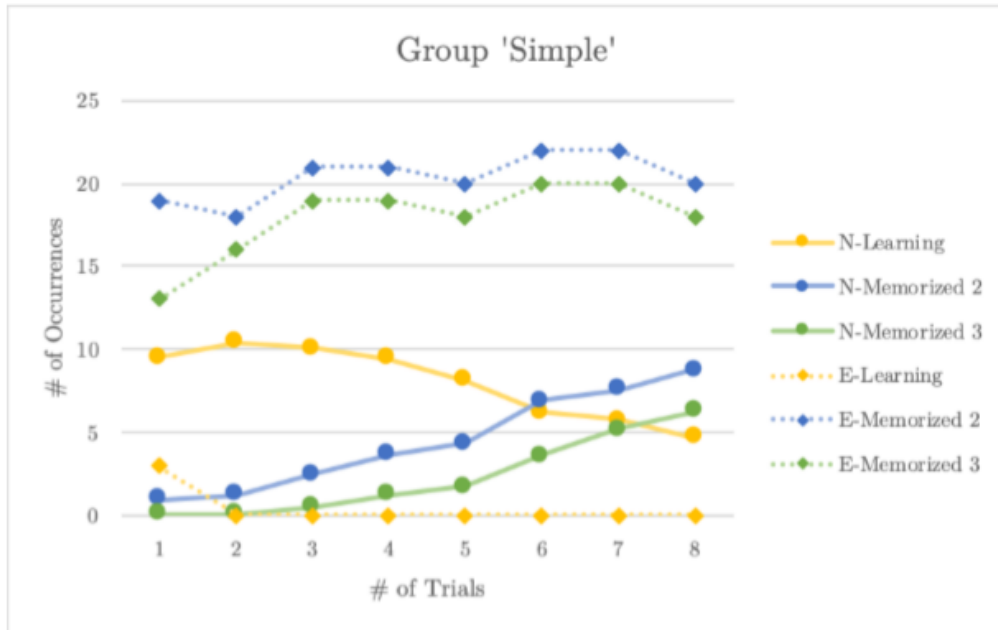


Figure 1.7: Gaze pattern occurrences for group ‘Simple’. The solid curves represent the novices (N) and the dotted curves the corresponding expert (E). The yellow curves represent the ‘Learning’ pattern, the blue ones ‘Memorized 2’ and the green ones ‘Memorized 3’.

The exact counts of pattern occurrences for the averaged gaze data of group ‘Simple’ are represented in Table 1.7.

Table 1.7: Gaze pattern occurrences for group ‘Simple’ - novices (N) and expert (E)

Pattern	N-Learning	N-Memorized 2	N-Memorized 3	E-Learning	E-Memorized 2	E-Memorized 3
Trial 1	9.5	1.0	0.1	3	19	13
Trial 2	10.4	1.3	0.1	0	18	16
Trial 3	10.1	2.5	0.5	0	21	19
Trial 4	9.4	3.7	1.3	0	21	19
Trial 5	8.1	4.4	1.8	0	20	18
Trial 6	6.2	6.9	3.6	0	22	20
Trial 7	5.8	7.6	5.2	0	22	20
Trial 8	4.7	8.8	6.3	0	20	18

### 1.3.2 Group ‘Complex’

The N-Learning pattern for the novices, with  $7.8 \pm 0.5$  occurrences on average, stays approximately constant over all 8 trials. The N-Memorized 2 increases

Figure A.2: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

steadily from the beginning (from trial 1 to trial 8), but the N-Learning and N-Memorized 2 curves never reach any intersection point. The N-Memorized 3 pattern does not occur once in any sequence and therefore, there is no intersection with N-Learning either (see Figure 1.8).

The expert of group ‘Complex’ on the other hand, shows a similar gaze behavior to the expert of group ‘Simple’. The E-Learning pattern occurs only once during trial 6 and remains on the level 0 for the remaining trials. By contrast, E-Memorized 2 and E-Memorized 3 are constantly present patterns over all 8 trials and occur in average  $18.4 \pm 1.1$  and  $16.4 \pm 1.1$  times per sequence (averaged over all trials) (see Figure 1.8).

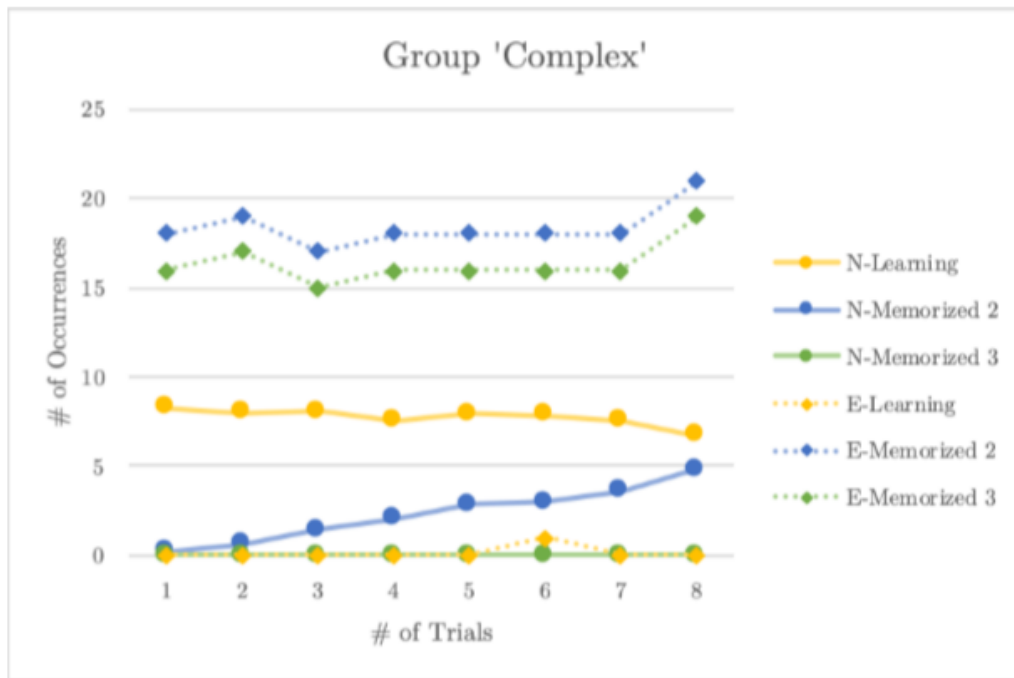


Figure 1.8: Gaze pattern occurrences for group ‘Complex’. The solid curves represent the novices (N) and the dotted curves the corresponding expert (E). The yellow curves represent the ‘Learning’ pattern, the blue ones ‘Memorized 2’ and the green ones ‘Memorized 3’.

The exact counts of pattern occurrences for the averaged gaze data of group ‘Complex’ are represented in Table 1.8.

Figure A.3: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

Table 1.8: Gaze pattern occurrences for group ‘Complex’ - novices (N) and expert (E)

Pattern	N-Learning	N-Memorized 2	N-Memorized 3	E-Learning	E-Memorized 2	E-Memorized 3
Trial 1	8.3	0.2	0.0	0	18	16
Trial 2	8.0	0.6	0.0	0	19	17
Trial 3	8.1	1.4	0.0	0	17	15
Trial 4	7.6	2.1	0.0	0	18	16
Trial 5	7.9	2.9	0.0	0	18	16
Trial 6	7.9	3.0	0.0	1	18	16
Trial 7	7.6	3.6	0.0	0	18	16
Trial 8	6.7	4.9	0.0	0	21	19

## 1.4 Dwell sequence comparison

The dwell sequences were compared individually for each participant (participant-internal sequence comparison as described in ??). The GLD was calculated for novices (N-GLD) of group ‘Simple’ and the related expert (E-GLD) only. For more detailed investigation, each dwell sequence per trial was divided into Part 1 and Part 2 (see ?? and ??) and compared individually. The GLD values based on the AOI-dwell sequences were used to compare trials participant-internally in order to deduce the individual learning behavior and quantify individuality.

### 1.4.1 Expert ‘Simple’

The GLD values of Part 1 and 2 of the expert of group ‘Simple’ are shown in Figure 1.9. The x-axis depicts the comparison of trial  $i$  versus trial  $i+1$  and the y-axis the resulting GLD scores. The expert, who has completely memorized the assembly of the plane, including building order, does not show a constant GLD value of 0 over the course of trial comparison. The single trial sequences (trial  $i$  and trial  $i+1$ ) do never coincide completely apart from sequences 4 and 5 for Part 1. The GLD values stay within a range of 0 and 4 for both parts (starting from trial 2 for Part 1 and from trial 3 for Part 2).

For gaze sequence comparison starting with ‘4 vs. 5’ to ‘7 vs. 8’ the averaged GLD for the expert is  $2.8 \pm 1.3$  for Part 1 and  $2.3 \pm 1.5$  for Part 2 (only the last 4 GLDs were considered for better comparison to the novices’ data in a second step in ??) (see Table 1.9).

Figure A.4: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

Table 1.9: GLD values for the expert ‘Simple’

GLD	Trial 1 vs. 2	Trial 2 vs. 3	Trial 3 vs. 4	Trial 4 vs. 5	Trial 5 vs. 6	Trial 6 vs. 7	Trial 7 vs. 8	Mean	StdDev
Part 1	8	2	2	2	4	1	4	2.8	1.3
Part 2	13	6	4	0	4	2	3	2.3	1.5



Figure 1.9: GLD values for expert ‘Simple’. The blue curve represents Part 1 of each trial and the green one Part 2 respectively.

### 1.4.2 Group ‘Simple’

The resulting GLD values of novices of group ‘Simple’ are illustrated in Figure 1.12 for Part 1 and in Figure 1.13 for Part 2. In both charts does the x-axis give the comparison of trial  $i$  versus trial  $i+1$  and the y-axis the resulting GLD scores.

Generally, the novices’ GLD values of Part 1 of group ‘Simple’ trend downwards from  $6.9 \pm 3.1$  to  $4.8 \pm 3.4$  (data averaged over all 14 novices, see Figure 1.10 or more detailed in Table 1.10). Participant 9 (P-9) is forming one exception (see Figure 1.12).

The same behavior can be observed in Figure 1.13 for Part 2 of Group ‘Simple’, although the GLD values are higher. Here, the downward trend is marked by  $27.8 \pm 15.8$  for ‘1 vs. 2’ and by  $8.9 \pm 4.1$  for ‘7 vs. 8’ averaged over all 14 novices (see Figure 1.11).

Figure A.5: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

Table 1.10: Averaged GLD values for group ‘Simple’ (Part1)

GLD	1 vs. 2	2 vs. 3	3 vs. 4	4 vs. 5	5 vs. 6	6 vs. 7	7 vs. 8
Mean	6.9	5.9	5.9	5.4	5.4	4.9	4.8
StdDev	3.2	2.6	2.7	2.2	2.2	4.1	3.4

Table 1.11: Averaged GLD values for group ‘Simple’. (Part 2)

GLD	1 vs. 2	2 vs. 3	3 vs. 4	4 vs. 5	5 vs. 6	6 vs. 7	7 vs. 8
Mean	27.8	17.5	15.6	10.1	11.1	8.4	8.9
StdDev	15.8	9.9	8.0	2.7	4.2	3.2	4.1

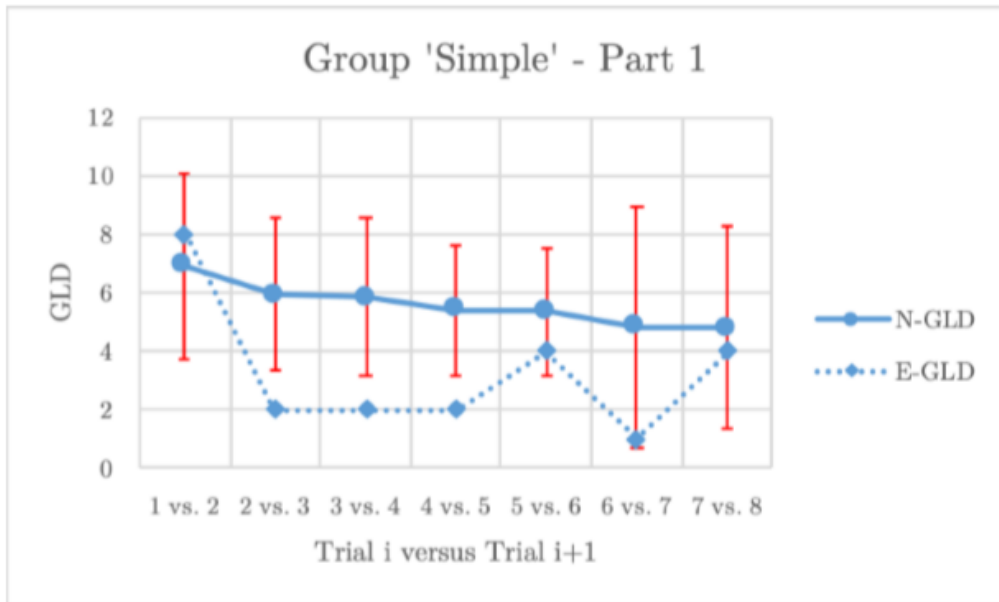


Figure 1.10: Averaged GLD values for group ‘Simple’. The solid curve represents the novices (N) and the dotted one the corresponding expert (E). The blue curves represent Part 1 of each assembly.

Figure A.6: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

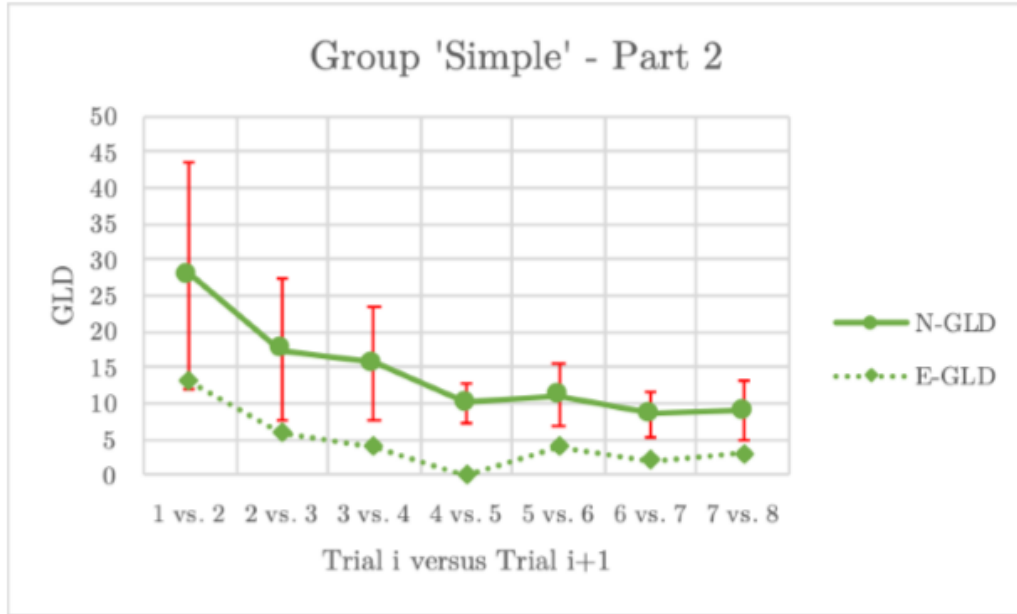


Figure 1.11: Averaged GLD values for group 'Simple'. The solid curve represents the novices (N) and the dotted one the corresponding expert (E). The green curves represent Part 2 of each assembly.

The GLD-curves (and therefore, the gaze AOI-visit sequences) for Part 1 and Part 2 of the novices of group 'Simple' in Figure 1.12 deviate from one another and do not show universal behavior. The exact GLD values for Part 1 and Part 2 of group 'Simple' are represented in Table 1.12 and Table 1.13 respectively.

Figure A.7: Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).



Table 1.12: GLD values for participants (P) of group ‘Simple’ (Part 1)

GLD	Trial 1 vs. 2	Trial 2 vs. 3	Trial 3 vs. 4	Trial 4 vs. 5	Trial 5 vs. 6	Trial 6 vs. 7	Trial 7 vs. 8
P-1	5	9	11	3	7	3	5
P-2	8	4	2	4	5	5	6
P-3	4	4	4	4	4	5	3
P-4	10	4	5	4	3	2	2
P-5	5	9	2	6	5	5	7
P-6	13	5	6	4	4	2	5
P-7	5	9	10	8	7	3	5
P-8	4	3	5	7	6	5	3
P-9	6	7	10	8	11	19	16
P-10	4	9	5	5	3	4	3
P-16	8	5	6	8	5	3	2
P-17	6	9	4	5	7	6	4
P-18	5	1	7	9	6	4	4
P-28	14	5	5	1	2	2	2

Table 1.13: GLD values for participants (P) of group ‘Simple’ (Part 2)

GLD	Trial 1 vs. 2	Trial 2 vs. 3	Trial 3 vs. 4	Trial 4 vs. 5	Trial 5 vs. 6	Trial 6 vs. 7	Trial 7 vs. 8
P-1	21	27	15	13	16	7	7
P-2	20	16	16	12	9	7	6
P-3	14	11	9	8	8	8	7
P-4	15	20	11	9	6	0	4
P-5	17	11	14	9	4	7	11
P-6	21	45	42	13	16	10	11
P-7	13	15	16	10	8	8	3
P-8	23	9	8	10	12	11	8
P-9	46	9	19	6	20	11	18
P-10	41	8	12	12	15	10	13
P-16	36	14	14	8	11	6	8
P-17	16	14	13	12	10	14	14
P-18	35	30	11	5	10	12	10
P-28	71	16	19	15	11	7	4

Figure A.8: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

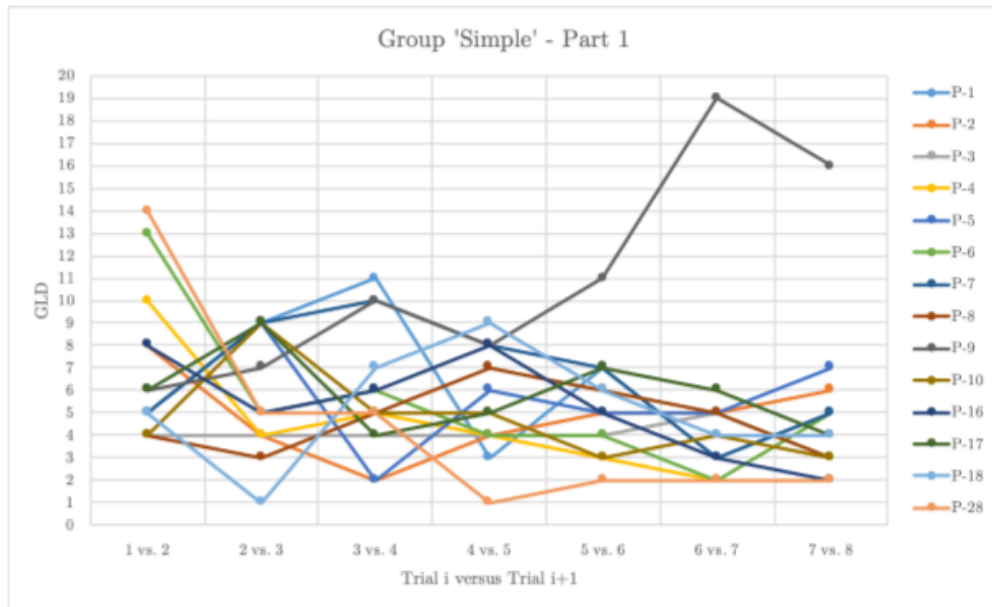


Figure 1.12: GLD values for participants (P) of group 'Simple' (Part 1)

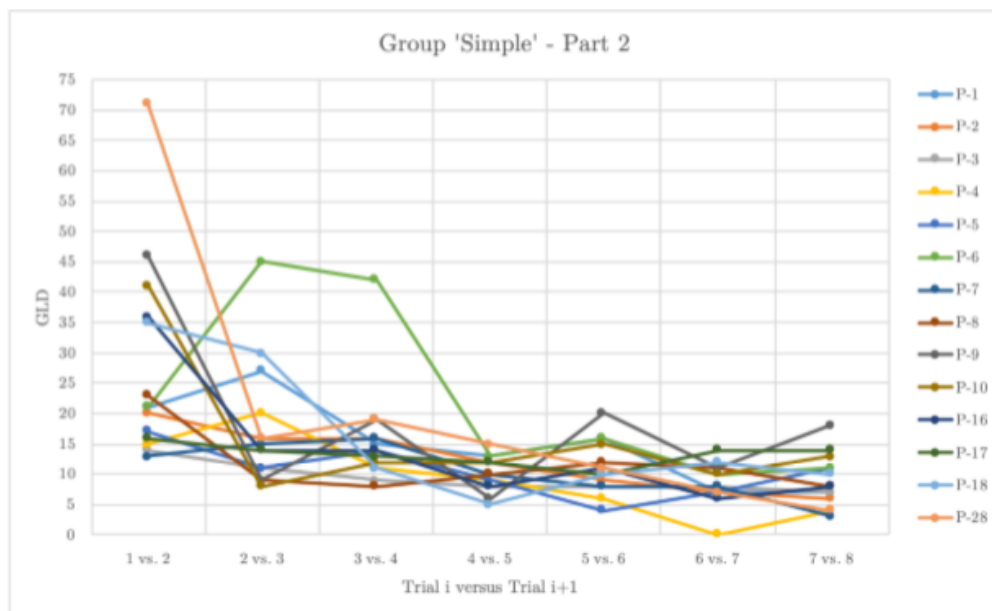


Figure 1.13: GLD values for participants (P) of group 'Simple' (Part 2)

Figure A.9: Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

For sequence comparison starting with ‘4 vs. 5’ to ‘7 vs. 8’ (only the last 4 GLD values were considered like for the expert of the respective group) the mean of GLD for Part 1 averaged over all 14 novices is  $5.1 \pm 1.4$  (see Table 1.14) and  $9.6 \pm 2.6$  for Part 2 (see Table 1.15).

Table 1.14: Averaged GLD values (from ‘4 vs. 5’ to ‘7 vs. 8’) for group ‘Simple’ (Part 1)

GLD	Mean	Stdev
P-1	4.5	1.7
P-2	5.0	0.7
P-3	4.0	0.7
P-4	2.8	0.8
P-5	5.8	0.8
P-6	3.8	1.1
P-7	5.8	1.9
P-8	5.3	1.5
P-9	13.5	4.3
P-10	3.8	0.8
P-16	4.5	2.3
P-17	5.5	1.1
P-18	5.8	2.0
P-28	1.8	0.4
<b>Average</b>	5.1	1.4

Figure A.10: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

Table 1.15: Averaged GLD values (from ‘4 vs. 5’ to ‘7 vs. 8’) for group ‘Simple’ (Part 2)

<b>GLD</b>	Mean	Stdev
P-1	10.8	3.9
P-2	8.5	2.3
P-3	7.8	0.4
P-4	4.8	3.3
P-5	7.8	2.6
P-6	12.5	2.3
P-7	7.3	2.6
P-8	10.3	1.5
P-9	13.8	5.6
P-10	12.5	1.8
P-16	8.3	1.8
P-17	12.5	1.7
P-18	9.3	2.6
P-28	9.3	4.1
<b>Average</b>	9.6	2.6

Figure A.11: Results of Celine Gianduzzo’s bachelor thesis in terms of AOI sequence analysis (Gianduzzo (2020)).

## B Source Code

The code is as well provided on GitHub and can be accessed with the following link:

<https://github.com/blivioo/Comparison-of-Task-Completion-using-Eye-Tracking-Recordings2-.git>

Code B.1: Header-Matlab script for sequence analysis

```
clear all;
clc;
%% To start the program provide/adjust the following
% :
%
% 1) If you'd like to analyse surgery-sequences:
% 1.1) Prepare 2 types of .txt files:
%      - The first type is to import the
%          Novice/Expert/Template-Data; as an
%          example take a look at
%          'Novice1.txt'; the layout and the
%          name(Name example: 3 Novice
%          textfilex --> Novice1.txt; Novice2.
%          txt; Novice3...)
%          MUST be the same as there; Layout: As
%          many
%          rows as surgeries performed by the
%          participant but MAX two
%          columns for characteristic partition
%          of each surgery;
%          Furthermore MAX 5 different templates
%          --> one per row
%
%      - The second type is to import the
%          performance
```

```

%           feedback steps; as an example take a
look at
%           'Crucial_Feedback_Cath_Steps1.txt';
the layout and the name MUST be the
%           same as there; MAX 10 rows as
performance feedback steps defined by the
%           user and MAX TWO columns; the first
column for description
%           and the second for the corresponding
abbreviation. If the
%           surgery you desire to analyse has TWO
CHARACTERISTIK PARTS
%           (P1,P2) provide one '
Crucial_Feedback_Cath_Steps1.txt' and one
%           'Crucial_Feedback_Cath_Steps2.txt'
for each part.
%
%   1.2)Score:
%           The initial Score of every
participant is set to be 10. As a
%           first step the edit-distances(only
levenshtein distance)
%           are analysed and the score is
calculated: there are
%           4 different "intervals"
%
%           first: edit-distance == 0 -->
InitialScore - 0
%           second: 1 <= edit-distance <= 2
--> InitialScore - 1
%           third: 3 <= edit-distance <= 5 -->
InitialScore - 3
%           fourth: 6 <= edit-distance <= 5
--> InitialScore - 5
%
%           Then, in a second step the user can
provide a weighting of
%           his PF-steps. The minimum weighting
is a deduction of -0.5
%           which one can weight up to four times
to get a deduction of -2
%           for more important steps. The program
can either find out if

```

```

%           a surgeon did undertake a specific
step or if he did NOT and
%           adjust the score respectively to the
user's input.
%           Furthermore the user can define steps
for which he is
%           interested in the number of
apperances.. For these
%           steps the user can define from which
count of appearances on he wants the
%           program to undertake a deduction.
%
%
% 2) If you'd like to analyse eye-tracking-
sequences(AOI-sequences):
% 2.1)Prepare 2 type of .txt file:
%           The names of the .txt files must be
as following:
%           - for Novices/Participants:
Participants_Type1.txt, where
%           type stands for number of different
experiments. These
%           files can contain as many rows as
novices recorded during
%           the specific experiment and as many
columns as parts*trials. If
%           a trial is separated in a first and
second part, the user
%           can provide this information with
the second question
%           asked from the program. AS AN
EXAMPLE TAKE A LOOK AT THE Examples-Files
%           FOLDER.
%           - The same as explained before must
be done for the Experts:
%           Experts_Type2.txt is a example for
the name of Expert-sequences from
%           experiment type2.
%           - For the Template file: Same layout
with a single sequence
%           per part or a whole experiment-set
with all trials,
%           which the user defines as to be the
ideal-sequence.

```

```

%           - As a last part the user needs to
provide a self-defined
%           text-file where one provides
crucial AOI-patterns or
%           single AOI's with the corresponding
abbreviations. In the first column
%           the user can place a description of
the abbreviation and in the
%           second column he can add the
corresponding abbreviations. The
%           program then searches as defined by
the user for
%           appearances, patterns or a
combination of the two. The user can define
%           if he wants the program to give the
number of total
%           appearances of the AOI-abbreviation
or if it shall simply
%           check if the AOI or AOI-pattern
appeared once in the
%           sequence.

%           - Store all the .txt files in the
DATA-folder in the
%           eye_track_seq-folder.
%
%   2.2)Change data-set: If the user wishes to
analyse a different
%           experiment-data-set, he has to change the
number at the end of the
%           file-names in the code. For example from
Participants_Type1.txt,
%           Experts_Type1.txt, Template_Type1.txt and
Find_Type1.txt for experiment 1
%           to Participants_Type2.txt, Experts_Type2.
txt, Template_Type2.txt and
%           Find_Type2.txt to analyse now data from
experiment two.
%
%% Ask user which part of the program he wishes to
use

```



```

%addpath(['/Volumes/1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis/
    Program/sequence_analysis/Cath_feedback_V2']);
addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\Cath_feedback_V2');
addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\eye_track_seq');

prompt = 'Would you like to analyse surgery-
    sequences or eye-tracking-sequences?(INPUT:1 or 2
    respectively)\n';

UserDesire = input(prompt);

if UserDesire == 1

    Cath_feedback_V2;

elseif UserDesire == 2

    usability_seq_comp2;

else disp('WRONG INPUT; START AGAIN');

end

```

## B.1 Surgical Sequence Analysis Code

All the functions used in the following scripts are attached at the end.

Code B.2: Matlab script for surgical sequence analysis

```

%% import sequences from all participants and store
    them in a cell array

addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\

```

```

    Program\sequence_analysis\Cath_feedback_V2\Data\
    Participant_Sequences');
addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\Cath_feedback_V2\Data\
    Procedure_Steps_Def');
addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\Cath_feedback_V2\Data\
    Templates');

%addpath(['Volumes/1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis/
    Program/sequence_analysis/Cath_feedback_V2/Data
    ']);
%addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\Cath_feedback_V2\Data\
    Participant_Sequences');
%addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\Cath_feedback_V2\Data\
    Participant_Sequences');

%find out what the size is of the users data-set:
prompt = 'How many Novices does your data-set
    involve?(INPUT:Number)\n';

numN = input(prompt);

prompt = 'How many Experts does your data-set
    involve?(INPUT:Number)\n';

numE = input(prompt);

%does the surgery the user wishes to analyse have
    two characteristics parts
%or are there only single sequences for the whole
    procedure?
prompt = 'Does the surgery you wish to analyse have
    two characteristic parts as defined by the
    seperation of left leg (LL) and right leg (RL)
    seen in the example Novice1.txt or not?(INPUT:2

```

```

        or 1 respectively)\n';

UserParts = input(prompt);
%import novice-data:
for i = 1:numN

    DataN(i,1) = {importfile2(sprintf('Novice%d.txt'
        , i),UserParts)};

end
%import experts-data:
for i = 1:numE

    DataE(i,1) = {importfile2(sprintf('Expert%d.txt'
        , i),UserParts)};

end

%% import Templates
prompt = 'How many different templates does your
    data-set involve?(INPUT:Number)\n';

numT = input(prompt);

for i = 1:numT

DataT(i,1) = {importfile2(sprintf('Template%d.txt',
    i),UserParts)};

end
%% Calculate score for each participant using
    Levenshtein-Dist., Damerau-Levenshtein-Dist. and
    user-defined feedback regulations

addpath('P:\04_Student_theses\1378
    _BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
    Program\sequence_analysis\Cath_feedback_V2\Data\
    Crucial_Feedback_Steps');

for i = 1:UserParts

DataPF(i,1) = {importfile4(sprintf('
    Crucial_Feedback_Cath_Steps%d.txt', i))};

```

```

end

%Score-Function which uses data-sets as inputs and
  Userinformation about his data-set and gives as
  output the scores and score-track:
[Scores,Scores_Track,ScoreN_Names,ScoreE_Names,
  TrackN_Names,TrackE_Names] = score_calc1(DataN,
  DataE, DataT, DataPF, UserParts, numN, numE);

%display results of score calculation:
for i = 1:numN

    disp(Scores.Novices.(ScoreN_Names(i,1)));
    disp(Scores_Track.Novices.(TrackN_Names(i,1)));

end

for i = 1:numE

    disp(Scores.Experts.(ScoreE_Names(i,1)));
    disp(Scores_Track.Experts.(TrackE_Names(i,1)));

end

```

Code B.3: Matlab script for surgeon score calculation (first score)

```

%% function for surgeon perf. feedback:
function [Scores,Scores_Track,ScoreN_Names,
  ScoreE_Names,TrackN_Names,TrackE_Names] =
  score_calc1(seqN, seqE, seqT, seqPF, UserInput,
  numN, numE)
%% Let user choose template if there is one provided
:

prompt = 'Which template of your template-file
  would you like to use?(INPUT:Number of specific
  row in template-file)\n';

TRowPF = input(prompt);

if (TRowPF == 1) && (UserInput == 2)

```

```
    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{1,2});
    TEMPL_P2 = convertStringsToChars(seqT
        {1,1}{1,3});

elseif (TRowPF == 2) && (UserInput == 2)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{2,2});
    TEMPL_P2 = convertStringsToChars(seqT
        {1,1}{2,3});

elseif (TRowPF == 3) && (UserInput == 2)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{3,2});
    TEMPL_P2 = convertStringsToChars(seqT
        {1,1}{3,3});

elseif (TRowPF == 4) && (UserInput == 2)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{4,2});
    TEMPL_P2 = convertStringsToChars(seqT
        {1,1}{4,3});

elseif (TRowPF == 5) && (UserInput == 2)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{5,2});
    TEMPL_P2 = convertStringsToChars(seqT
        {1,1}{5,3});

elseif (TRowPF == 1) && (UserInput == 1)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{1,2});

elseif (TRowPF == 2) && (UserInput == 1)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{2,2});
```

```

elseif (TRowPF == 3) && (UserInput == 1)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{3,2});

elseif (TRowPF == 4) && (UserInput == 1)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{4,2});

elseif (TRowPF == 5) && (UserInput == 1)

    TEMPL_P1 = convertStringsToChars(seqT
        {1,1}{5,2});

end

%% assign data to number of participant-variables(
    Novices/Experts) to calculate edit-distances:

for j = 1:numN
    %dynamic creation of variables needed for the
        score calculation according
    %to users data size:
    %generate variable needed
    my_variablesN = sprintf('seq_N%d', j);
    %store it in struct and assign data of novices
        seqN
    NovicesS(1).(my_variablesN) = seqN{j,1};
    %generate string with all struct variable names
        to be able to use loops
    %for calculation
    NSeqNames(j,1) = convertCharsToStrings(
        my_variablesN);

    %variables for respectively levenshtein distance
        storage for part 1 of
    %surgery
    my_variablesN = sprintf('L_P1_N%d', j);
    NovicesD1(1).(my_variablesN) = {};
    N_NamesD1(j,1) = convertCharsToStrings(
        my_variablesN);

```

```

%variables for respectively damerau-levenshtein
    distance storage part 1
%of surgery
my_variablesN = sprintf('D_L_P1_N%d', j);
NovicesD1(1).(my_variablesN) = {};
N_NamesD1(j+numN,1) = convertCharsToStrings(
    my_variablesN);

%same for part2
my_variablesN = sprintf('L_P2_N%d', j);
NovicesD2(1).(my_variablesN) = {};
N_NamesD2(j,1) = convertCharsToStrings(
    my_variablesN);

%same for part2
my_variablesN = sprintf('D_L_P2_N%d', j);
NovicesD2(1).(my_variablesN) = {};
N_NamesD2(j+numN,1) = convertCharsToStrings(
    my_variablesN);

%struct to store all the distances to display
    data in overall table per
%participant
my_variablesN = sprintf('allD_N%d', j);
AllDN(1).(my_variablesN) = [];
AllDN_names(j,1) = convertCharsToStrings(
    my_variablesN);

%table structs to store tables containing
    distances:
my_variablesN = sprintf('TN%d', j);
my_variablesN_L = sprintf('L_DN%d', j);
my_variablesN_DL = sprintf('DL_DN%d', j);
AllTN(1).(my_variablesN) = [];
AllTN_names(j,1) = convertCharsToStrings(
    my_variablesN);
AllTN_names(j,2) = convertCharsToStrings(
    my_variablesN_L);
AllTN_names(j,3) = convertCharsToStrings(
    my_variablesN_DL);

%struct to save scores of novices:

```

```

my_variablesN = sprintf('ScoresN%d', j);
ScoresN(1).(my_variablesN) = [];
ScoresN_names(j,1) = convertCharsToStrings(
    my_variablesN);

%struct to save scores_track of novices:
my_variablesN = sprintf('Scores_TrackN%d', j);
Scores_TrackN(1).(my_variablesN) = [];
Scores_TrackN_names(j,1) = convertCharsToStrings(
    my_variablesN);

%tables stored in struct with the scores and
    score-track to display
%results in a organised way:
my_variablesN = sprintf('T_ScoreN%d', j);
T_ScoreN(1).(my_variablesN) = [];
T_ScoreN_names(j,1) = convertCharsToStrings(
    my_variablesN);

my_variablesN = sprintf('T_TrackN%d', j);
T_TrackN(1).(my_variablesN) = [];
T_TrackN_names(j,1) = convertCharsToStrings(
    my_variablesN);

end

%same for expert-data, see explanation above:
for j = 1:numE

    my_variablesE = sprintf('seq_E%d', j);
    ExpertsS(1).(my_variablesE) = seqE{j,1};
    ESeqNames(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('L_P1_E%d', j);
    ExpertsD1(1).(my_variablesE) = {};
    E_NamesD1(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('D_L_P1_E%d', j);
    ExpertsD1(1).(my_variablesE) = {};
    E_NamesD1(j+numE,1) = convertCharsToStrings(
        my_variablesE);

```



```

my_variablesE = sprintf('L_P2_E%d', j);
ExpertsD2(1).(my_variablesE) = {};
E_NamesD2(j,1) = convertCharsToStrings(
    my_variablesE);

my_variablesE = sprintf('D_L_P2_E%d', j);
ExpertsD2(1).(my_variablesE) = {};
E_NamesD2(j+numE,1) = convertCharsToStrings(
    my_variablesE);

my_variablesE = sprintf('allD_E%d', j);
AllDE(1).(my_variablesE) = [];
AllDE_names(j,1) = convertCharsToStrings(
    my_variablesE);

my_variablesE = sprintf('TE%d', j);
my_variablesE_L = sprintf('L_DE%d', j);
my_variablesE_DL = sprintf('DL_DE%d', j);
AllTE(1).(my_variablesE) = [];
AllTE_names(j,1) = convertCharsToStrings(
    my_variablesE);
AllTE_names(j,2) = convertCharsToStrings(
    my_variablesE_L);
AllTE_names(j,3) = convertCharsToStrings(
    my_variablesE_DL);

my_variablesE = sprintf('ScoresE%d', j);
ScoresE(1).(my_variablesE) = [];
ScoresE_names(j,1) = convertCharsToStrings(
    my_variablesE);

my_variablesE = sprintf('Scores_TrackE%d', j);
Scores_TrackE(1).(my_variablesE) = [];
Scores_TrackE_names(j,1) = convertCharsToStrings(
    my_variablesE);

my_variablesE = sprintf('T_ScoreE%d', j);
T_ScoreE(1).(my_variablesE) = [];
T_ScoreE_names(j,1) = convertCharsToStrings(
    my_variablesE);

my_variablesE = sprintf('T_TrackE%d', j);

```

```

    T_TrackE(1).(my_variablesE) = [];
    T_TrackE_names(j,1) = convertCharsToStrings(
        my_variablesE);

end

%% calculate edit-distances for single partitioned
    surgery

if UserInput == 1

while (true)

while (true)

    prompt = 'Would you like to calculate the
        Levenshtein-Dist. and the Damerau-Levenshtein-
        Dist.?(INPUT:yes or no)\n';
    UserDistance = input(prompt,'s');

    if strcmp(UserDistance,'no')
%%Levenshtein-dist.:

        disp('
            -----
        ');
        disp('The sequences of the novices and experts
            were compared with the beforehand chosen
            template');
        disp('and these were the corresponding Levenshtein
            distances (and Damerau-Levenshtein distances):
        ');

%%Novices:
        for i = 1:numN

            for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

                %calculate levenshtein dist. for each
                    sequence and store it in
                %distance struct NovicesD1 where D1 stands
                    for distances in part1
                %of surgery containing no further

```

```

        characteristic parts:
NovicesD1(1).(N_NamesD1(i,1)){j,1} =
    L_distance(TEMPL_P1,
    convertStringsToChars(NovicesS(1).(
    NSeqNames(i,1)){j,2}));

end

%put distances in display struct where cell
%is changed to a table
%with the row and column names for this
%calculation:
AllDN(1).(AllDN_names(i,1)){1,1} = cell2mat(
    NovicesD1(1).(N_NamesD1(i,1)).');

AllTN(1).(AllTN_names(i,1)) = cell2table(
    AllDN(1).(AllDN_names(i,1)), '
VariableNames', {'P1'}, 'RowNames', {
    convertStringsToChars(AllTN_names(i,2))}
    );
disp(AllTN(1).(AllTN_names(i,1)));

end

%Experts: (same as above but for different type of
%participant)
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        %L_distance needs character vectors as
        %input, therefore the
        %sequences must be converted to it:
ExpertsD1(1).(E_NamesD1(i,1)){j,1} =
    L_distance(TEMPL_P1,
    convertStringsToChars(ExpertsS(1).(
    ESeqNames(i,1)){j,2}));

    end

    AllDE(1).(AllDE_names(i,1)){1,1} = cell2mat(
        ExpertsD1(1).(E_NamesD1(i,1)).');

    AllTE(1).(AllTE_names(i,1)) = cell2table(

```

```

        AllDE(1).(AllDE_names(i,1)), '
        VariableNames', {'P1'}, 'RowNames', {
        convertStringsToChars(AllTE_names(i,2))})
    ;
    disp(AllTE(1).(AllTE_names(i,1)));

end

%% calculate Levenshtein-Dist. and Damerau-
Levenshtein-Dist.

elseif strcmp(UserDistance, 'yes')

    disp('
    -----
    ');
    disp('The sequences of the novices and experts
    were compared with the beforehand chosen
    template');
    disp('and these were the corresponding Levenshtein
    distances (and Damerau-Levenshtein distances):
    ');

%Fist levenshtein dist. is calculated
%Novices
for i = 1:numN

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        NovicesD1(1).(N_NamesD1(i,1)){j,1} =
            L_distance(TEMPL_P1,
            convertStringsToChars(NovicesS(1).(
            NSeqNames(i,1)){j,2}));

    end

end

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

```

```

        ExpertsD1(1).(E_NamesD1(i,1)){j,1} =
            L_distance(TEMPL_P1,
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(i,1)){j,2})));

    end

end

%in a second step damerau-levenshtein dist. is
    calculated:
%Novices:
    for i = 1:numN

        for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

            NovicesD1(1).(N_NamesD1(i+numN,1)){j,1} =
                damerau_levenshtein(TEMPL_P1,
                    convertStringsToChars(NovicesS(1).(
                        NSeqNames(i,1)){j,2})));

        end

        %assign distances to struct with all
            distances and generate table
        %with row and column names according to
            participant names:
        AllDN(1).(AllDN_names(i,1)){1,1} = cell2mat(
            NovicesD1(1).(N_NamesD1(i,1)).');
        AllDN(1).(AllDN_names(i,1)){2,1} = cell2mat(
            NovicesD1(1).(N_NamesD1(i+numN,1)).');

        AllTN(1).(AllTN_names(i,1)) = cell2table(
            AllDN(1).(AllDN_names(i,1)), '
            VariableNames', {'P1'}, ...
            'RowNames', {convertStringsToChars(
                AllTN_names(i,2)),
                convertStringsToChars(AllTN_names(i,3))});
        disp(AllTN(1).(AllTN_names(i,1)));

    end

```

```

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        ExpertsD1(1).(E_NamesD1(i+numE,1)){j,1} =
            damerau_levenshtein(TEMPL_P1,
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(i,1)){j,2}));

    end

    AllDE(1).(AllDE_names(i,1)){1,1} = cell2mat(
        ExpertsD1(1).(E_NamesD1(i,1)).');
    AllDE(1).(AllDE_names(i,1)){2,1} = cell2mat(
        ExpertsD1(1).(E_NamesD1(i+numE,1)).');

    AllTE(1).(AllTE_names(i,1)) = cell2table(
        AllDE(1).(AllDE_names(i,1)), '
        VariableNames', {'P1'}, 'RowNames', ...
        {convertStringsToChars(AllTE_names(i,2)),
            convertStringsToChars(AllTE_names(i
            ,3))});
    disp(AllTE(1).(AllTE_names(i,1)));

end

%if user types something else than yes or no the
    loop starts from the
%beginning, ensuring no unintended error occurs:
else disp('WRONG INPUT');

    break;

end

break;

end

break;

```

```

end

%% calculate distances for two partitioned surgery
%%here all the calculations have to be undertaken for
    part1 and part2:
elseif UserInput == 2

while (true)

while (true)

    prompt = 'Would you like to calculate the
              Levenshtein-Dist. and the Damerau-Levenshtein-
              Dist.?(INPUT:yes or no)\n';
    UserDistance = input(prompt,'s');

    if strcmp(UserDistance,'no')
%%Levenshtein-dist.:

        disp('
            -----
            ');
        disp('The sequences of the novices and experts
              were compared with the beforehand chosen
              template');
        disp('and these were the corresponding Levenshtein
              distances (and Damerau-Levenshtein distances):
            ');

%%Novices:
        for i = 1:numN

            for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

                NovicesD1(1).(N_NamesD1(i,1)){j,1} =
                    L_distance(TEMPL_P1,
                        convertStringsToChars(NovicesS(1).(
                            NSeqNames(i,1)){j,2}));
                %part2 distances get assigned to ...D2
                %struct, the rest stays the
                %same as above for a single partitioned
                %surgery
                NovicesD2(1).(N_NamesD2(i,1)){j,1} =

```

```

        L_distance(TEMPL_P2,
        convertStringsToChars(NovicesS(1).(
        NSeqNames(i,1)){j,3}));

    end

    % now in column 1 distances from part 1 get
    % assigned and in column 2
    % distances from part 2:
    AllDN(1).(AllDN_names(i,1)){1,1} = cell2mat(
        NovicesD1(1).(N_NamesD1(i,1)).');
    AllDN(1).(AllDN_names(i,1)){1,2} = cell2mat(
        NovicesD2(1).(N_NamesD2(i,1)).');

    AllTN(1).(AllTN_names(i,1)) = cell2table(
        AllDN(1).(AllDN_names(i,1)), '
        VariableNames', {'P1' 'P2'}, 'RowNames',
        {convertStringsToChars(AllTN_names(i,2))
        });
    disp(AllTN(1).(AllTN_names(i,1)));

end

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        ExpertsD1(1).(E_NamesD1(i,1)){j,1} =
            L_distance(TEMPL_P1,
            convertStringsToChars(ExpertsS(1).(
            ESeqNames(i,1)){j,2}));

        ExpertsD2(1).(E_NamesD2(i,1)){j,1} =
            L_distance(TEMPL_P2,
            convertStringsToChars(ExpertsS(1).(
            ESeqNames(i,1)){j,3}));

    end

    AllDE(1).(AllDE_names(i,1)){1,1} = cell2mat(
        ExpertsD1(1).(E_NamesD1(i,1)).');
    AllDE(1).(AllDE_names(i,1)){1,2} = cell2mat(

```



```

        ExpertsD2(1).(E_NamesD2(i,1)).');

    AllTE(1).(AllTE_names(i,1)) = cell2table(
        AllDE(1).(AllDE_names(i,1)), '
        VariableNames', {'P1' 'P2'}, 'RowNames',
        {convertStringsToChars(AllTE_names(i,2))
        });
    disp(AllTE(1).(AllTE_names(i,1)));

end

%% calculate Levenshtein-Dist. and Damerau-
    Levenshtein-Dist.

elseif strcmp(UserDistance, 'yes')

    disp('
    -----
    ');
    disp('The sequences of the novices and experts
        were compared with the beforehand chosen
        template');
    disp('and these were the corresponding Levenshtein
        distances (and Damerau-Levenshtein distances):
    ');

%Novices
for i = 1:numN

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        NovicesD1(1).(N_NamesD1(i,1)){j,1} =
            L_distance(TEMPL_P1,
            convertStringsToChars(NovicesS(1).(
            NSeqNames(i,1)){j,2}));

        NovicesD2(1).(N_NamesD2(i,1)){j,1} =
            L_distance(TEMPL_P2,
            convertStringsToChars(NovicesS(1).(
            NSeqNames(i,1)){j,3}));

    end
end

```

```

end

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        ExpertsD1(1).(E_NamesD1(i,1)){j,1} =
            L_distance(TEMPL_P1,
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(i,1)){j,2}));

        ExpertsD2(1).(E_NamesD2(i,1)){j,1} =
            L_distance(TEMPL_P2,
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(i,1)){j,3}));

    end

end

%Novices:
for i = 1:numN

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        % damerau levenshtein dist. get assigned to
        % the damerau-levenshtein
        % arrays called by their names N_NamesD1(i+
        % numN,1) corresponding
        % to part 1 or 2 they belong to:
        NovicesD1(1).(N_NamesD1(i+numN,1)){j,1} =
            damerau_levenshtein(TEMPL_P1,
                convertStringsToChars(NovicesS(1).(
                    NSeqNames(i,1)){j,2}));

        NovicesD2(1).(N_NamesD2(i+numN,1)){j,1} =
            damerau_levenshtein(TEMPL_P2,
                convertStringsToChars(NovicesS(1).(
                    NSeqNames(i,1)){j,3}));

    end

end

```

```

%for displayment all levenshtein distances
    per participant and per surgery
%undertaken get assigned to row 1 and
    damerau-lev. dist. to row 2;
%the differentiation between part 1 and part
    2 is ensured by
%putting part1 distances in column 1 and
    part 2 distances in column
%2. as a example the first number in (1,1)
    stands for the
%levenshtein distance for part 1 in the
    first surgery, and (1,2)
%for the distance for part 2.
%the same structure is also applied to the
    damerau-lev. distances:
AllDN(1).(AllDN_names(i,1)){1,1} = cell2mat(
    NovicesD1(1).(N_NamesD1(i,1)).');
AllDN(1).(AllDN_names(i,1)){1,2} = cell2mat(
    NovicesD2(1).(N_NamesD2(i,1)).');
AllDN(1).(AllDN_names(i,1)){2,1} = cell2mat(
    NovicesD1(1).(N_NamesD1(i+numN,1)).');
AllDN(1).(AllDN_names(i,1)){2,2} = cell2mat(
    NovicesD2(1).(N_NamesD2(i+numN,1)).');

AllTN(1).(AllTN_names(i,1)) = cell2table(
    AllDN(1).(AllDN_names(i,1)), '
VariableNames', {'P1' 'P2'}, ...
    'RowNames', {convertStringsToChars(
        AllTN_names(i,2)),
        convertStringsToChars(AllTN_names(i,3)
        )});
disp(AllTN(1).(AllTN_names(i,1)));

end

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        ExpertsD1(1).(E_NamesD1(i+numE,1)){j,1} =
            damerau_levenshtein(TEMPL_P1,
            convertStringsToChars(ExpertsS(1).(

```

```

        ESeqNames(i,1)){j,2}));

    ExpertsD2(1).(E_NamesD2(i+numE,1)){j,1} =
        damerau_levenshtein(TEMPL_P2,
        convertStringsToChars(ExpertsS(1).(
        ESeqNames(i,1)){j,3})));

end

    AllDE(1).(AllDE_names(i,1)){1,1} = cell2mat(
        ExpertsD1(1).(E_NamesD1(i,1)).');
    AllDE(1).(AllDE_names(i,1)){1,2} = cell2mat(
        ExpertsD2(1).(E_NamesD2(i,1)).');
    AllDE(1).(AllDE_names(i,1)){2,1} = cell2mat(
        ExpertsD1(1).(E_NamesD1(i+numE,1)).');
    AllDE(1).(AllDE_names(i,1)){2,2} = cell2mat(
        ExpertsD2(1).(E_NamesD2(i+numE,1)).');

    AllTE(1).(AllTE_names(i,1)) = cell2table(
        AllDE(1).(AllDE_names(i,1)), '
        VariableNames', {'P1' 'P2'}, 'RowNames',
        ...
        {convertStringsToChars(AllTE_names(i,2)),
        convertStringsToChars(AllTE_names(i
        ,3))});
    disp(AllTE(1).(AllTE_names(i,1)));

end

else disp('WRONG INPUT');

    break;

end

    break;

end

    break;

end
end

```

```

end
%% Calculate final Score of Novices/Experts:

%to understand the score calculation read the first
description in
%sequence_analysis.m

disp('
-----
');
disp('For the calculation of the final score, only
the Levenshtein distance is considered.')
InScore = 100;

%1) Edit-Distance interval for first score calc.:
%there are three intervals defined:

if UserInput == 1

%Novices:
for i = 1:numN
    %check into which interval the respective
    distance calculated above falls
    %and assign starting score according to the
    define deductions per
    %interval
    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        if (AllDN(1).(AllDN_names(i,1)){1,1}(1,j) >= 1)
            && (AllDN(1).(AllDN_names(i,1)){1,1}(1,j) <=
                2)

                ScoresN(1).(ScoresN_names(i,1))(1,j) =
                    InScore - 10;
                %keep track of what caused the score
                deduction with scores_track..
                %to be able to understand in the end how the
                score was calculated:
                Scores_TrackN(1).(Scores_TrackN_names(i,1))
                    {1,j} = (-10);

        elseif (AllDN(1).(AllDN_names(i,1)){1,1}(1,j) ==
            0)

```

```

        ScoresN(1).(ScoresN_names(i,1))(1,j) =
            InScore;
        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,j} = 0;

elseif (AllDN(1).(AllDN_names(i,1)){1,1}(1,j) >=
        3) && (AllDN(1).(AllDN_names(i,1)){1,1}(1,j)
        <= 5)

        ScoresN(1).(ScoresN_names(i,1))(1,j) =
            InScore - 30;
        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,j} = (-30);

elseif (AllDN(1).(AllDN_names(i,1)){1,1}(1,j) >=
        6)

        ScoresN(1).(ScoresN_names(i,1))(1,j) =
            InScore - 40;
        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,j} = (-40);

    end

end

end

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        if (AllDE(1).(AllDE_names(i,1)){1,1}(1,j) >= 1)
            && (AllDE(1).(AllDE_names(i,1)){1,1}(1,j) <=
                2)

            ScoresE(1).(ScoresE_names(i,1))(1,j) =
                InScore - 10;
            Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {1,j} = (-10);
        end
    end
end

```

```

elseif (AllDE(1).(AllDE_names(i,1)){1,1}(1,j) ==
    0)

    ScoresE(1).(ScoresE_names(i,1))(1,j) =
        InScore;
    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {1,j} = 0;

elseif (AllDE(1).(AllDE_names(i,1)){1,1}(1,j) >=
    3) && (AllDE(1).(AllDE_names(i,1)){1,1}(1,j)
    <= 5)

    ScoresE(1).(ScoresE_names(i,1))(1,j) =
        InScore - 30;
    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {1,j} = (-30);

elseif (AllDE(1).(AllDE_names(i,1)){1,1}(1,j) >=
    6)

    ScoresE(1).(ScoresE_names(i,1))(1,j) =
        InScore - 40;
    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {1,j} = (-40);

end

end

end

%do the same twice if the surgery contains part1 and
part2:
elseif UserInput == 2

%Novices:
for i = 1:numN

    k = 1;

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        if (AllDN(1).(AllDN_names(i,1)){1,1}(1,j) >= 1)

```

```

    && (AllDN(1).(AllDN_names(i,1)){1,1}(1,j)
    <= 2)

    ScoresN(1).(ScoresN_names(i,1))(1,k) =
        InScore - 10;

    Scores_TrackN(1).(Scores_TrackN_names(i,1))
        {1,k} = (-10);

    %use k as an independend running variable to
    %assign infromation for
    %each surgery-part, part 1 and 2 next to each
    %other in array before
    %switching to next surgery:
    k = k+1;

elseif (AllDN(1).(AllDN_names(i,1)){1,1}(1,j)
    == 0)

    ScoresN(1).(ScoresN_names(i,1))(1,k) =
        InScore;

    Scores_TrackN(1).(Scores_TrackN_names(i,1))
        {1,k} = 0;

    k = k+1;

elseif (AllDN(1).(AllDN_names(i,1)){1,1}(1,j)
    >= 3) && (AllDN(1).(AllDN_names(i,1))
    {1,1}(1,j) <= 5)

    ScoresN(1).(ScoresN_names(i,1))(1,k) =
        InScore - 30;

    Scores_TrackN(1).(Scores_TrackN_names(i,1))
        {1,k} = (-30);

    k = k+1;

elseif (AllDN(1).(AllDN_names(i,1)){1,1}(1,j)
    >= 6)

    ScoresN(1).(ScoresN_names(i,1))(1,k) =

```



```

        InScore = 40;

        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,k} = (-40);

        k = k+1;

    end

    if (AllDN(1).(AllDN_names(i,1)){1,2}(1,j) >= 1)
        && (AllDN(1).(AllDN_names(i,1)){1,2}(1,j)
            <= 2)

        ScoresN(1).(ScoresN_names(i,1))(1,k) =
            InScore - 10;

        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,k} = (-10);

        k = k+1;

    elseif (AllDN(1).(AllDN_names(i,1)){1,2}(1,j)
        == 0)

        ScoresN(1).(ScoresN_names(i,1))(1,k) =
            InScore;

        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,k} = 0;

        k = k+1;

    elseif (AllDN(1).(AllDN_names(i,1)){1,2}(1,j)
        >= 3) && (AllDN(1).(AllDN_names(i,1))
            {1,2}(1,j) <= 5)

        ScoresN(1).(ScoresN_names(i,1))(1,k) =
            InScore - 30;

        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {1,k} = (-30);

        k = k+1;

```

```

elseif (AllDN(1).(AllDN_names(i,1)){1,2}(1,j)
    >= 6)

    ScoresN(1).(ScoresN_names(i,1))(1,k) =
        InScore - 40;

    Scores_TrackN(1).(Scores_TrackN_names(i,1))
        {1,k} = (-40);

    k = k+1;

end

end

end

%Experts:
for i = 1:numE

    k = 1;

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        if (AllDE(1).(AllDE_names(i,1)){1,1}(1,j) >= 1)
            && (AllDE(1).(AllDE_names(i,1)){1,1}(1,j)
                <= 2)

            ScoresE(1).(ScoresE_names(i,1))(1,k) =
                InScore - 10;

            Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {1,k} = (-10);

            k = k+1;

            elseif (AllDE(1).(AllDE_names(i,1)){1,1}(1,j)
                == 0)

            ScoresE(1).(ScoresE_names(i,1))(1,k) =
                InScore;

```

```

        Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {1,k} = 0;

        k = k+1;

elseif (AllDE(1).(AllDE_names(i,1)){1,1}(1,j)
    >= 3) && (AllDE(1).(AllDE_names(i,1))
    {1,1}(1,j) <= 5)

    ScoresE(1).(ScoresE_names(i,1))(1,k) =
        InScore - 30;

    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {1,k} = (-30);

    k = k+1;

elseif (AllDE(1).(AllDE_names(i,1)){1,1}(1,j)
    >= 6)

    ScoresE(1).(ScoresE_names(i,1))(1,k) =
        InScore - 40;

    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {1,k} = (-40);

    k = k+1;

end

if (AllDE(1).(AllDE_names(i,1)){1,2}(1,j) >= 1)
    && (AllDE(1).(AllDE_names(i,1)){1,2}(1,j)
    <= 2)

    ScoresE(1).(ScoresE_names(i,1))(1,k) =
        InScore - 10;

    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {1,k} = (-10);

    k = k+1;

elseif (AllDE(1).(AllDE_names(i,1)){1,2}(1,j)

```

```

        == 0)

        ScoresE(1).(ScoresE_names(i,1))(1,k) =
            InScore;

        Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {1,k} = 0;

        k = k+1;

elseif (AllDE(1).(AllDE_names(i,1)){1,2}(1,j)
        >= 3) && (AllDE(1).(AllDE_names(i,1))
        {1,2}(1,j) <= 5)

        ScoresE(1).(ScoresE_names(i,1))(1,k) =
            InScore - 30;

        Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {1,k} = (-30);

        k = k+1;

elseif (AllDE(1).(AllDE_names(i,1)){1,2}(1,j)
        >= 6)

        ScoresE(1).(ScoresE_names(i,1))(1,k) =
            InScore - 40;

        Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {1,k} = (-40);

        k = k+1;

    end

end

end

end

%2) User-provided crucial Perf.-Feedb. cases for
    rest of score:

```

```
if UserInput == 2
    disp('
    -----
    ');
    disp('Do you wish to use a predefined PF-template
    for the catheterization-score-calculation');
    disp('which contains crucial steps with already
    defined weighting factors for this surgery?(
    INPUT:yes or no)');

    %since the program was designed to analyse
    different surgeries but the
    %initial design was done to compare surgeons-
    vessel catheterization
    %skills before a minimal invasive cardiovascular
    surgery, there is a
    %already predefined "user-provided" crucial perf.
    feedback template which
    %the user can now choose to use or not:
    prompt = '';
    UseTemp = input(prompt, 's');

    if strcmp(UseTemp, 'no')

        score_calc1_twopartitioned;

    elseif strcmp(UseTemp, 'yes')

        score_calc1_catheterization;

    end

elseif UserInput == 1

    score_calc1_singlepartitioned;

end

Scores(1).Novices = T_ScoreN;
Scores(1).Experts = T_ScoreE;
Scores_Track.Novices = T_TrackN;
Scores_Track.Experts = T_TrackE;
```

```

ScoreN_Names = T_ScoreN_names;
ScoreE_Names = T_ScoreE_names;
TrackN_Names = T_TrackN_names;
TrackE_Names = T_TrackE_names;

end

```

Code B.4: Matlab script for catheterization-score-calculation with self defined crucial performance feedback steps and weighting factors (second score)

```

%% Score_calculation for catheterization surgery:

%for explanation of variables and loops see comments
in
%score_calc1_twopartitioned.m

numPF1 = 5;

numPF2 = 5;

for j =1:numPF1

    my_variablesPF1 = sprintf('PF1_%d', j);
    PF1(1).(my_variablesPF1) = string(seqPF{1,1}(j,2)
    );
    PF1_names(j,1) = convertCharsToStrings(
        my_variablesPF1);

    my_variablesWf1 = sprintf('Wf1_%d', j);
    Wf1(1).(my_variablesWf1) = [];
    Wf1_names(j,1) = convertCharsToStrings(
        my_variablesWf1);

end

for j =1:numPF2

    my_variablesPF2 = sprintf('PF2_%d', j);
    PF2(1).(my_variablesPF2) = string(seqPF{2,1}(j,2)
    );
    PF2_names(j,1) = convertCharsToStrings(
        my_variablesPF2);

```

```
my_variablesWf2 = sprintf('Wf2_%d', j);
Wf2(1).(my_variablesWf2) = [];
Wf2_names(j,1) = convertCharsToStrings(
    my_variablesWf2);

end

%First PF1:
numNotThere1 = 1;

NotThere1(1,1) = 3;

numThere1 = numPF1 - numNotThere1;

There1(1,1) = 1;
There1(2,1) = 2;
There1(3,1) = 4;
There1(4,1) = 5;

NumWdif1 = 3;
NumWin1 = numPF1 - NumWdif1;

Win = 4;

RowWdif1(1,1) = 1;
RowWdif1(2,1) = 3;
RowWdif1(3,1) = 4;

RowWdif1(1,2) = 4;
RowWdif1(2,2) = 4;
RowWdif1(3,2) = 2;

for i = 1:NumWdif1
    Wf1(1).(Wf1_names((RowWdif1(i,1)),1)) = RowWdif1(i
        ,2)*Win;
end

NumApp1 = 1;

RowApp1(1,1) = 2;

CountStart1 = 1;
```

```
RowWin1(1,1) = 2;
RowWin1(2,1) = 5;

for i = 1:NumWin1
    Wf1(1).(Wf1_names((RowWin1(i,1)),1)) = Win;
end

%Second PF2:

numNotThere2 = 2;

NotThere2(1,1) = 1;
NotThere2(2,1) = 3;

numThere2 = numPF2 - numNotThere2;

There2(1,1) = 2;
There2(2,1) = 4;
There2(3,1) = 5;

NumWdif2 = 4;
NumWin2 = numPF2 - NumWdif2;

Win = 4;

RowWdif2(1,1) = 1;
RowWdif2(2,1) = 3;
RowWdif2(3,1) = 4;
RowWdif2(4,1) = 5;

RowWdif2(1,2) = 8;
RowWdif2(2,2) = 3;
RowWdif2(3,2) = 2;
RowWdif2(4,2) = 4;

for i = 1:NumWdif2
    Wf2(1).(Wf2_names((RowWdif2(i,1)),1)) = RowWdif2(i,2)*Win;
end

NumApp2 = 1;
```



```

RowApp2(1,1) = 2;

CountStart2 = 1;

RowWin2(1,1) = 2;

    for i = 1:NumWin2
        Wf2(1).(Wf2_names((RowWin2(i,1)),1)) = Win;
    end

%calculate score with this information:
%Novices:

for i = 1:numN

    k4 = 1;

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        if numNotThere1 >= 1

            k3 = 1;

            %Part1:
            for k1 = 1:numThere1

                for k2 = 1:NumApp1

                    if ~strcmp(PF1(1).(PF1_names(There1(k1,1),1)),
                        PF1(1).(PF1_names(RowApp1(k2,1),1)))

                        if contains(NovicesS(1).(NSeqNames(i,1)){j
                            ,2}, PF1(1).(PF1_names(There1(k1,1),1)))

                            ScoresN(1).(ScoresN_names(i,1))(1,k4) =
                                ScoresN(1).(ScoresN_names(i,1))(1,k4) -
                                (Wf1(1).(Wf1_names(There1(k1,1),1)));

                            Scores_TrackN(1).(Scores_TrackN_names(i,1))
                                {k3+1,k4} = seqPF{1,1}{There1(k1,1),1};
                            %Scores_TrackN(1).(Scores_TrackN_names(i,1)
                                ){k3+1,k4} = PF1(1).(PF1_names(There1(k1
                                ,1),1));

```

```

        k3 = k3+1;

    end

elseif strcmp(PF1(1).(PF1_names(There1(k1,1)
    ,1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

    if contains(NovicesS(1).(NSeqNames(i,1)){j
        ,2}, PF1(1).(PF1_names(There1(k1,1),1)))

        count = length(strfind(string(NovicesS(1).(
            NSeqNames(i,1)){j,2}), PF1(1).(PF1_names
                (There1(k1,1),1))));
        if count > CountStart1
            CountMult1 = count - CountStart1;
            ScoresN(1).(ScoresN_names(i,1))(1,k4) =
                ScoresN(1).(ScoresN_names(i,1))(1,k4) -
                (Wf1(1).(Wf1_names(There1(k1,1),1))*
                    CountMult1);

            Scores_TrackN(1).(Scores_TrackN_names(i,1))
                {k3+1,k4} = seqPF{1,1}{There1(k1,1),1};
            %Scores_TrackN(1).(Scores_TrackN_names(i,1)
                ){k3+1,k4} = PF1(1).(PF1_names(There1(k1
                    ,1),1));

            k3 = k3+1;

        end

    end

end

end

end

for k1 = 1:numNotThere1

    for k2 = 1:NumApp1

```

```

if ~strcmp(PF1(1).(PF1_names(NotThere1(k1,1),1)
), PF1(1).(PF1_names(RowApp1(k2,1),1)))

if ~contains(NovicesS(1).(NSeqNames(i,1)){j
,2}, PF1(1).(PF1_names(NotThere1(k1,1),1)))

ScoresN(1).(ScoresN_names(i,1))(1,k4) =
    ScoresN(1).(ScoresN_names(i,1))(1,k4) - (
    Wf1(1).(Wf1_names(NotThere1(k1,1),1)));

%Scores_TrackN(1).(Scores_TrackN_names(i,1))
{k3+1,k4} = PF1(1).(PF1_names(NotThere1(
k1,1),1));
Scores_TrackN(1).(Scores_TrackN_names(i,1)){
k3+1,k4} = seqPF{1,1}{NotThere1(k1,1),1};

k3 = k3+1;

end

elseif strcmp(PF1(1).(PF1_names(NotThere1(k1,1)
,1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

if ~contains(NovicesS(1).(NSeqNames(i,1)){j
,2}, PF1(1).(PF1_names(NotThere1(k1,1),1)))

count = length(strfind(string(NovicesS(1).(
NSeqNames(i,1)){j,2}), PF1(1).(PF1_names(
NotThere1(k1,1),1))));
if count > CountStart1
CountMult1 = count - CountStart1;
ScoresN(1).(ScoresN_names(i,1))(1,k4) =
    ScoresN(1).(ScoresN_names(i,1))(1,k4) - (
    Wf1(1).(Wf1_names(NotThere1(k1,1),1))*
    CountMult1);

%Scores_TrackN(1).(Scores_TrackN_names(i,1))
{k3+1,k4} = PF1(1).(PF1_names(NotThere1(
k1,1),1));
Scores_TrackN(1).(Scores_TrackN_names(i,1)){
k3+1,k4} = seqPF{1,1}{NotThere1(k1,1),1};

```

```

        k3 = k3+1;

    end

end

end

end

end

end

if numNotThere2 >= 1

k3 = 1;

%Part2:
for k1 = 1:numThere2

    for k2 = 1:NumApp2

        if ~strcmp(PF2(1).(PF2_names(There2(k1,1),1))
            , PF2(1).(PF2_names(RowApp2(k2,1),1)))

            if contains(NovicesS(1).(NSeqNames(i,1)){j
                ,3}, PF2(1).(PF2_names(There2(k1,1),1)))

                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
                    ScoresN(1).(ScoresN_names(i,1))(1,k4+1)
                    - (Wf2(1).(Wf2_names(There2(k1,1),1)));

                %Scores_TrackN(1).(Scores_TrackN_names(i,1)
                    ){k3+1,k4+1} = PF2(1).(PF2_names(There2(
                    k1,1),1));
                Scores_TrackN(1).(Scores_TrackN_names(i,1))
                    {k3+1,k4+1} = seqPF{2,1}{There2(k1,1)
                    ,1};

            k3 = k3+1;

```

```

        end

elseif strcmp(PF2(1).(PF2_names(There2(k1,1),1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

if contains(NovicesS(1).(NSeqNames(i,1)){j,3}, PF2(1).(PF2_names(There2(k1,1),1)))

count = length(strfind(string(NovicesS(1).(NSeqNames(i,1)){j,3}), PF2(1).(PF2_names(There2(k1,1),1))));
if count > CountStart2
CountMult2 = count - CountStart2;
ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
    ScoresN(1).(ScoresN_names(i,1))(1,k4+1)
    - (Wf2(1).(Wf2_names(There2(k1,1),1))*
    CountMult2);

%Scores_TrackN(1).(Scores_TrackN_names(i,1)
    ){k3+1,k4+1} = PF2(1).(PF2_names(There2(
    k1,1),1));
Scores_TrackN(1).(Scores_TrackN_names(i,1)
    ){k3+1,k4+1} = seqPF{2,1}{There2(k1,1)
    ,1};

k3 = k3+1;

end

end

end

end

end

for k1 = 1:numNotThere2

for k2 = 1:NumApp2

if ~strcmp(PF2(1).(PF2_names(NotThere2(k1,1),1)

```

```

    ), PF2(1).(PF2_names(RowApp2(k2,1),1)))

    if ~contains(NovicesS(1).(NSeqNames(i,1)){j
        ,3}, PF2(1).(PF2_names(NotThere2(k1,1),1)))

        ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
            ScoresN(1).(ScoresN_names(i,1))(1,k4+1) -
            (Wf2(1).(Wf2_names(NotThere2(k1,1),1)));

        %Scores_TrackN(1).(Scores_TrackN_names(i,1))
        {k3+1,k4+1} = PF2(1).(PF2_names(NotThere2
            (k1,1),1));
        Scores_TrackN(1).(Scores_TrackN_names(i,1)){
            k3+1,k4+1} = seqPF{2,1}{NotThere2(k1,1)
                ,1};

        k3 = k3+1;

    end

elseif strcmp(PF2(1).(PF2_names(NotThere2(k1,1)
    ,1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

    if ~contains(NovicesS(1).(NSeqNames(i,1)){j
        ,3}, PF2(1).(PF2_names(NotThere2(k1,1),1)))

        count = length(strfind(string(NovicesS(1).(
            NSeqNames(i,1)){j,3}), PF2(1).(PF2_names(
                NotThere2(k1,1),1))));
        if count > CountStart2
            CountMult2 = count - CountStart2;
            ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) -
                (Wf2(1).(Wf2_names(NotThere2(k1,1),1))*
                    CountMult2);

            %Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {k3+1,k4+1} = PF2(1).(PF2_names(NotThere2
                (k1,1),1));
            Scores_TrackN(1).(Scores_TrackN_names(i,1))
                {k3+1,k4+1} = seqPF{2,1}{NotThere2(k1,1)
                    ,1};
        end
    end
end

```

```

        k3 = k3+1;

        end

    end

    end

    end

    end

    end

    end

    k4 = k4+2;

    end

end

%Experts:
for i = 1:numE

    k4 = 1;

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        if numNotThere1 >= 1

            k3 = 1;

            %Part1:
            for k1 = 1:numThere1

                for k2 = 1:NumApp1

                    if ~strcmp(PF1(1).(PF1_names(There1(k1,1),1))
                        , PF1(1).(PF1_names(RowApp1(k2,1),1)))

                        if contains(ExpertsS(1).(ESeqNames(i,1)){j
                            ,2}, PF1(1).(PF1_names(There1(k1,1),1)))

```

```

    ScoresE(1).(ScoresE_names(i,1))(1,k4) =
        ScoresE(1).(ScoresE_names(i,1))(1,k4) -
        (Wf1(1).(Wf1_names(There1(k1,1),1)));

    %Scores_TrackE(1).(Scores_TrackE_names(i,1)
        ){k3+1,k4} = PF1(1).(PF1_names(There1(k1
            ,1),1));
    Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {k3+1,k4} = seqPF{1,1}{There1(k1,1),1};

    k3 = k3+1;

end

elseif strcmp(PF1(1).(PF1_names(There1(k1,1)
    ,1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

if contains(ExpertsS(1).(ESeqNames(i,1)){j
    ,2}, PF1(1).(PF1_names(There1(k1,1),1)))

    count = length(strfind(string(ExpertsS(1).(
        ESeqNames(i,1)){j,2}), PF1(1).(PF1_names
            (There1(k1,1),1))));
    if count > CountStart1
        CountMult1 = count - CountStart1;
        ScoresE(1).(ScoresE_names(i,1))(1,k4) =
            ScoresE(1).(ScoresE_names(i,1))(1,k4) -
            (Wf1(1).(Wf1_names(There1(k1,1),1))*
                CountMult1);

        %Scores_TrackE(1).(Scores_TrackE_names(i,1)
            ){k3+1,k4} = PF1(1).(PF1_names(There1(k1
                ,1),1));
        Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {k3+1,k4} = seqPF{1,1}{There1(k1,1),1};

        k3 = k3+1;

    end

end
end

```



```

        end

    end

end

for k1 = 1:numNotThere1

    for k2 = 1:NumApp1

        if ~strcmp(PF1(1).(PF1_names(NotThere1(k1,1),1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

            if ~contains(ExpertsS(1).(ESeqNames(i,1)){j,2}, PF1(1).(PF1_names(NotThere1(k1,1),1)))

                ScoresE(1).(ScoresE_names(i,1))(1,k4) =
                    ScoresE(1).(ScoresE_names(i,1))(1,k4) - (
                        Wf1(1).(Wf1_names(NotThere1(k1,1),1)));

                %Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {k3+1,k4} = PF1(1).(PF1_names(NotThere1(
                    k1,1),1));
                Scores_TrackE(1).(Scores_TrackE_names(i,1)){
                    k3+1,k4} = seqPF{1,1}{NotThere1(k1,1),1};

                k3 = k3+1;

            end

            elseif strcmp(PF1(1).(PF1_names(NotThere1(k1,1),1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

                if ~contains(ExpertsS(1).(ESeqNames(i,1)){j,2}, PF1(1).(PF1_names(NotThere1(k1,1),1)))

                    count = length(strfind(string(ExpertsS(1).(
                        ESeqNames(i,1)){j,2}), PF1(1).(PF1_names(
                            NotThere1(k1,1),1))));
                    if count > CountStart1
                        CountMult1 = count - CountStart1;
                        ScoresE(1).(ScoresE_names(i,1))(1,k4) =

```

```

        ScoresE(1).(ScoresE_names(i,1))(1,k4) - (
        Wf1(1).(Wf1_names(NotThere1(k1,1),1))*
        CountMult1);

    %Scores_TrackE(1).(Scores_TrackE_names(i,1))
    {k3+1,k4} = PF1(1).(PF1_names(NotThere1(
    k1,1),1));
    Scores_TrackE(1).(Scores_TrackE_names(i,1))
    {k3+1,k4} = seqPF{1,1}{NotThere1(k1,1)
    ,1};

    k3 = k3+1;

end

end

end

end

end

end

if numNotThere2 >= 1

    k3 = 1;

    %Part2:
    for k1 = 1:numThere2

        for k2 = 1:NumApp2

            if ~strcmp(PF2(1).(PF2_names(There2(k1,1),1))
            , PF2(1).(PF2_names(RowApp2(k2,1),1)))

                if contains(ExpertsS(1).(ESeqNames(i,1)){j
                ,3}, PF2(1).(PF2_names(There2(k1,1),1)))

                    ScoresE(1).(ScoresE_names(i,1))(1,k4+1) =
                    ScoresE(1).(ScoresE_names(i,1))(1,k4+1)

```

```

        - (Wf2(1).(Wf2_names(There2(k1,1),1)));

%Scores_TrackE(1).(Scores_TrackE_names(i,1)
    ){k3+1,k4+1} = PF2(1).(PF2_names(There2(
    k1,1),1));
Scores_TrackE(1).(Scores_TrackE_names(i,1))
    {k3+1,k4+1} = seqPF{2,1}{There2(k1,1)
    ,1};

k3 = k3+1;

end

elseif strcmp(PF2(1).(PF2_names(There2(k1,1)
    ,1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

if contains(ExpertsS(1).(ESeqNames(i,1)){j
    ,3}, PF2(1).(PF2_names(There2(k1,1),1)))

count = length(strfind(string(ExpertsS(1).(
    ESeqNames(i,1)){j,3}), PF2(1).(PF2_names
    (There2(k1,1),1))));
if count > CountStart2
CountMult2 = count - CountStart2;
ScoresE(1).(ScoresE_names(i,1))(1,k4+1) =
    ScoresE(1).(ScoresE_names(i,1))(1,k4+1)
    - (Wf2(1).(Wf2_names(There2(k1,1),1))*
    CountMult2);

%Scores_TrackE(1).(Scores_TrackE_names(i,1)
    ){k3+1,k4+1} = PF2(1).(PF2_names(There2(
    k1,1),1));
Scores_TrackE(1).(Scores_TrackE_names(i,1))
    {k3+1,k4+1} = seqPF{2,1}{There2(k1,1)
    ,1};

k3 = k3+1;

end

end

```

```

        end

    end

end

for k1 = 1:numNotThere2

    for k2 = 1:NumApp2

        if ~strcmp(PF2(1).(PF2_names(NotThere2(k1,1),1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

            if ~contains(ExpertsS(1).(ESeqNames(i,1)){j,3}, PF2(1).(PF2_names(NotThere2(k1,1),1)))

                ScoresE(1).(ScoresE_names(i,1))(1,k4+1) =
                    ScoresE(1).(ScoresE_names(i,1))(1,k4+1) -
                    (Wf2(1).(Wf2_names(NotThere2(k1,1),1)));

                %Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {k3+1,k4+1} = PF2(1).(PF2_names(NotThere2(k1,1),1));
                Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {k3+1,k4+1} = seqPF{2,1}{NotThere2(k1,1),1};

                k3 = k3+1;

            end

            elseif strcmp(PF2(1).(PF2_names(NotThere2(k1,1),1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

                if ~contains(ExpertsS(1).(ESeqNames(i,1)){j,3}, PF2(1).(PF2_names(NotThere2(k1,1),1)))

                    count = length(strfind(string(ExpertsS(1).(ESeqNames(i,1)){j,3}), PF2(1).(PF2_names(NotThere2(k1,1),1))));
                    if count > CountStart2
                        CountMult2 = count - CountStart2;
                        ScoresE(1).(ScoresE_names(i,1))(1,k4+1) =

```

```

        ScoresE(1).(ScoresE_names(i,1))(1,k4+1) -
            (Wf2(1).(Wf2_names(NotThere2(k1,1),1))*
            CountMult2);

        %Scores_TrackE(1).(Scores_TrackE_names(i,1))
        {k3+1,k4+1} = PF2(1).(PF2_names(NotThere2
            (k1,1),1));
        Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {k3+1,k4+1} = seqPF{2,1}{NotThere2(k1,1)
            ,1};

        k3 = k3+1;

    end

end

end

end

end

end

k4 = k4+2;

end

end

%Put Scores and ScoreTrack in tables:
%Novices:
for j = 1:numN

    numSurgeries = size(NovicesS.(NSeqNames(j,1)),1);
    Variables = {};
    Variables(1,1) = {'S1P1'};
    Variables(1,2) = {'S1P2'};
    colcount = 1;

    for i = 1:numSurgeries

```

```

    my_variables1 = sprintf('S%dP1', i);
    Variables11(1,i) = {my_variables1};

end

for i = 1:numSurgeries

    my_variables2 = sprintf('S%dP2', i);
    Variables22(1,i) = {my_variables2};

end

for i = 1:2:(2*numSurgeries)

    Variables(1,i) = Variables11(1,colcount);
    Variables(1,i+1) = Variables22(1,colcount);

    colcount = colcount + 1;

end

DeductionReason = {};
colcountT = 1;
for i = 1:size(Scores_TrackN.(Scores_TrackN_names(
    j,1)),1)

    DeductionReason(1,1) = {'LevDistIntDeduction'
        };

    if (size(Scores_TrackN.(Scores_TrackN_names(j
        ,1)),1) > 1) && (i > 1)

        DeductionReason(1,i) = {sprintf('
            DeductionReason%d',i-1)};

    end

end

Variables = string(Variables);

T_ScoreN.(T_ScoreN_names(j,1)) = array2table(

```

```

        ScoresN.(ScoresN_names(j,1)), 'VariableNames',
        Variables, 'RowNames', {sprintf('ScoresN%d:',j)
    });

    T_TrackN.(T_TrackN_names(j,1)) = array2table(
        Scores_TrackN.(Scores_TrackN_names(j,1)), '
        VariableNames', Variables, 'RowNames',
        DeductionReason);

end

%Experts:
for j = 1:numE

    numSurgeries = size(ExpertsS.(ESeqNames(j,1)),1);
    Variables = {};
    Variables(1,1) = {'S1P1'};
    Variables(1,2) = {'S1P2'};
    colcount = 1;

    for i = 1:numSurgeries

        my_variables1 = sprintf('S%dP1', i);
        Variables11(1,i) = {my_variables1};

    end

    for i = 1:numSurgeries

        my_variables2 = sprintf('S%dP2', i);
        Variables22(1,i) = {my_variables2};

    end

    for i = 1:2:(2*numSurgeries)

        Variables(1,i) = Variables11(1,colcount);
        Variables(1,i+1) = Variables22(1,colcount);

        colcount = colcount + 1;

    end
end

```

```

DeductionReason = {};
colcountT = 1;
for i = 1:size(Scores_TrackE.(Scores_TrackE_names(
    j,1)),1)

    DeductionReason(1,1) = {'LevDistIntDeduction'
        };

    if (size(Scores_TrackE.(Scores_TrackE_names(j
        ,1)),1) > 1) && (i > 1)

        DeductionReason(1,i) = {sprintf('
            DeductionReason%d',i-1)};

    end

end

Variables = string(Variables);

T_ScoreE.(T_ScoreE_names(j,1)) = array2table(
    ScoresE.(ScoresE_names(j,1)), 'VariableNames',
    Variables, 'RowNames', {sprintf('ScoresE%d:',j)
    });

T_TrackE.(T_TrackE_names(j,1)) = array2table(
    Scores_TrackE.(Scores_TrackE_names(j,1)), '
    VariableNames', Variables, 'RowNames',
    DeductionReason);

end

```

Code B.5: Matlab script for single partitioned surgery-score-calculation with user defined crucial feedback steps and weighting factors (second score)

```

%% Score_calculation with single partitioned surgery
:

prompt = 'How many feedback steps/definitions does
your PF data-file have?(INPUT: Number of rows in
Crucial_Feedback_Cath_Steps-file)\n';

```



```

numPF = input(prompt);

for j =1:numPF

    my_variablesPF = sprintf('PF_%d', j);
    PF(1).(my_variablesPF) = string(seqPF{1,1}(j,2));
    PF_names(j,1) = convertCharsToStrings(
        my_variablesPF);

    my_variablesWf = sprintf('Wf_%d', j);
    Wf(1).(my_variablesWf) = [];
    Wf_names(j,1) = convertCharsToStrings(
        my_variablesWf);

end

disp(seqPF{1,1});
prompt = 'From your PF-steps how many shall the
    programm treat as: if they do NOT appear execute
    a deduction in the score?(INPUT:Row number)\n';
numNotThere = input(prompt);

if numNotThere ~= 0

for i = 1:numNotThere

prompt = 'Type the row number of these specific
    steps.(INPUT:Row number)\n';
NotThere(i,1) = input(prompt);

end

end

numThere = numPF - numNotThere;
if numNotThere ~= 0

for i = 1:numThere

prompt = 'Type the row number of the specific steps
    you want the program to treat as: if they do
    appear execute a deduction in the score.(INPUT:

```

```

        Row number top to bottom of file)\n';
There(i,1) = input(prompt);

end

end

prompt = 'From your PF-steps how many do you wish to
weight stronger than -0.5?(INPUT:number)\n';
NumWdif = input(prompt);
NumWIn = numPF - NumWdif;

disp('As a first step type the row number of the
specific step. Then the weighting factor, which
can max be 4.(INPUT:Number of row, factor for
weighting)');
WIn = 0.5;

for i = 1:NumWdif

prompt = 'Row number: ';
RowWdif(i,1) = input(prompt);

prompt = 'Weight factor of step: ';
RowWdif(i,2) = input(prompt);
Wf(1).(Wf_names((RowWdif(i,1)),1)) = RowWdif(i,2)*
WIn;

end

prompt = 'Of how many PF-steps would you like to
know the number of appearances?(INPUT:number)\n';
NumApp = input(prompt);
disp('Now type the row number of these steps
respectively.(INPUT:Number of row)');

for i = 1:NumApp

prompt = 'Row number: ';
RowApp(i,1) = input(prompt);

end

```

```

prompt = 'After which count do you want the program
to deduct -0.5 per further count?(INPUT:number)\n
';
CountStart = input(prompt);

for i = 1:NumWIn

    prompt = 'From your PF-steps which ones do you wish
to weight with -0.5?(INPUT:Row number)\n';
    RowWIn(i,1) = input(prompt);
    Wf(1).(Wf_names((RowWIn(i,1)),1)) = WIn;

end

%calculate score with this information:
%Novices:
for i = 1:numN

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)

        if numNotThere == 0

            k3 = 1;

            for k1 = 1:numPF

                for k2 = 1:NumApp

                    if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
                        PF_names(RowApp(k2,1),1)))

                        if contains(NovicesS(1).(NSeqNames(i,1)){j
                            ,2}, PF(1).(PF_names(k1,1)))

                            ScoresN(1).(ScoresN_names(i,1))(1,j) =
                                ScoresN(1).(ScoresN_names(i,1))(1,j) - (
                                    Wf(1).(Wf_names(k1,1)));

                            Scores_TrackN(1).(Scores_TrackN_names(i,1)){
                                k3+1,j} = PF(1).(PF_names(k1,1));

                            k3 = k3+1;

```

```

        end

elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
    PF_names(RowApp(k2,1),1)))

    if contains(NovicesS(1).(NSeqNames(i,1)){j
        ,2}, PF(1).(PF_names(k1,1)))

        count = length(strfind(string(NovicesS(1).(
            NSeqNames(i,1)){j,2}), PF(1).(PF_names(k1
                ,1))));

        if count > CountStart

            CountMult = count - CountStart;

            ScoresN(1).(ScoresN_names(i,1))(1,j) =
                ScoresN(1).(ScoresN_names(i,1))(1,j)
                - (Wf(1).(Wf_names(k1,1))*CountMult);

            Scores_TrackN(1).(Scores_TrackN_names(i
                ,1)){k3+1,j} = PF(1).(PF_names(k1,1))
                ;

            k3 = k3+1;

        end

    end

end

end

end

elseif numNotThere >= 1

k3 = 1;

    for k1 = 1:numThere

```

```

for k2 = 1:NumApp

    if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
        PF_names(RowApp(k2,1),1)))

        if contains(NovicesS(1).(NSeqNames(i,1)){j
            ,2}, PF(1).(PF_names(There(k1,1),1)))

            ScoresN(1).(ScoresN_names(i,1))(1,j) =
                ScoresN(1).(ScoresN_names(i,1))(1,j) - (
                    Wf(1).(Wf_names(There(k1,1),1)));

            Scores_TrackN(1).(Scores_TrackN_names(i,1))
                {k3+1,j} = PF(1).(PF_names(There(k1,1)
                    ,1));

            k3 = k3+1;

        end

    elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
        PF_names(RowApp(k2,1),1)))

        if contains(NovicesS(1).(NSeqNames(i,1)){j
            ,2}, PF(1).(PF_names(There(k1,1),1)))

            count = length(strfind(string(NovicesS(1).(
                NSeqNames(i,1)){j,2}), PF(1).(PF_names(
                    There(k1,1),1))));

            if count > CountStart

                CountMult = count - CountStart;

                ScoresN(1).(ScoresN_names(i,1))(1,j) =
                    ScoresN(1).(ScoresN_names(i,1))(1,j) -
                    (Wf(1).(Wf_names(There(k1,1),1))*
                        CountMult);

                Scores_TrackN(1).(Scores_TrackN_names(i
                    ,1)){k3+1,j} = PF(1).(PF_names(There(
                        k1,1),1));

```

```

        k3 = k3+1;

    end

end

end

end

end

for k1 = 1:numNotThere

    for k2 = 1:NumApp

        if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
            PF_names(RowApp(k2,1),1)))

            if ~contains(NovicesS(1).(NSeqNames(i,1)){j
                ,2}, PF(1).(PF_names(NotThere(k1,1),1)))

                ScoresN(1).(ScoresN_names(i,1))(1,j) =
                    ScoresN(1).(ScoresN_names(i,1))(1,j) - (
                        Wf(1).(Wf_names(NotThere(k1,1),1)));

                Scores_TrackN(1).(Scores_TrackN_names(i,1)){
                    k3+1,j} = PF(1).(PF_names(NotThere(k1,1)
                        ,1));

                k3 = k3+1;

            end

            elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
                PF_names(RowApp(k2,1),1)))

                if ~contains(NovicesS(1).(NSeqNames(i,1)){j
                    ,2}, PF(1).(PF_names(NotThere(k1,1),1)))

                    count = length(strfind(string(NovicesS(1).(

```

```
NSeqNames(i,1)){j,2}), PF(1).(PF_names(
NotThere(k1,1),1))));

if count > CountStart

    CountMult = count - CountStart;

    ScoresN(1).(ScoresN_names(i,1))(1,j) =
        ScoresN(1).(ScoresN_names(i,1))(1,j)
        - (Wf(1).(Wf_names(NotThere(k1,1),1))
            *CountMult);

    Scores_TrackN(1).(Scores_TrackN_names(i
        ,1)){k3+1,j} = PF(1).(PF_names(
        NotThere(k1,1),1));

    k3 = k3+1;

end

end

end

end

end

end

end

end

%Experts:
for i = 1:numE

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        if numNotThere == 0
```

```

k3 = 1;

for k1 = 1:numPF

    for k2 = 1:NumApp

        if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
            PF_names(RowApp(k2,1),1)))

            if contains(ExpertsS(1).(ESeqNames(i,1)){j
                ,2}, PF(1).(PF_names(k1,1)))

                ScoresE(1).(ScoresE_names(i,1))(1,j) =
                    ScoresE(1).(ScoresE_names(i,1))(1,j) - (
                        Wf(1).(Wf_names(k1,1)));

                Scores_TrackE(1).(Scores_TrackE_names(i,1)){
                    k3+1,j} = PF(1).(PF_names(k1,1));

                k3 = k3+1;

            end

            elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
                PF_names(RowApp(k2,1),1)))

                if contains(ExpertsS(1).(ESeqNames(i,1)){j
                    ,2}, PF(1).(PF_names(k1,1)))

                    count = length(strfind(string(ExpertsS(1).(
                        ESeqNames(i,1)){j,2}), PF(1).(PF_names(k1
                            ,1))));
                    if count > CountStart
                        CountMult = count - CountStart;
                        ScoresE(1).(ScoresE_names(i,1))(1,j) =
                            ScoresE(1).(ScoresE_names(i,1))(1,j) - (
                                Wf(1).(Wf_names(k1,1))*CountMult);
                        Scores_TrackE(1).(Scores_TrackE_names(i,1))
                            {k3+1,j} = PF(1).(PF_names(k1,1));
                        k3 = k3+1;
                    end

                end

            end

```



```

        end

    end

end

elseif numNotThere >= 1

k3 = 1;

for k1 = 1:numThere

    for k2 = 1:NumApp

        if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
            PF_names(RowApp(k2,1),1)))

            if contains(ExpertsS(1).(ESeqNames(i,1)){j
                ,2}, PF(1).(PF_names(There(k1,1),1)))

                ScoresE(1).(ScoresE_names(i,1))(1,j) =
                    ScoresE(1).(ScoresE_names(i,1))(1,j) - (
                        Wf(1).(Wf_names(There(k1,1),1)));

                Scores_TrackE(1).(Scores_TrackE_names(i,1))
                    {k3+1,j} = PF(1).(PF_names(There(k1,1)
                        ,1));

                k3 = k3+1;

            end

            elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
                PF_names(RowApp(k2,1),1)))

                if contains(ExpertsS(1).(ESeqNames(i,1)){j
                    ,2}, PF(1).(PF_names(There(k1,1),1)))

                    count = length(strfind(string(ExpertsS(1).(
                        ESeqNames(i,1)){j,2}), PF(1).(PF_names(
                            There(k1,1),1))));

```

```

        if count > CountStart
            CountMult = count - CountStart;
            ScoresE(1).(ScoresE_names(i,1))(1,j) =
                ScoresE(1).(ScoresE_names(i,1))(1,j) -
                (Wf(1).(Wf_names(There(k1,1),1))*
                CountMult);

            Scores_TrackE(1).(Scores_TrackE_names(i,1)
                ){k3+1,j} = PF(1).(PF_names(There(k1,1)
                ,1));

            k3 = k3+1;

        end

    end

end

end

end

for k1 = 1:numNotThere

    for k2 = 1:NumApp

        if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
            PF_names(RowApp(k2,1),1)))

            if ~contains(ExpertsS(1).(ESeqNames(i,1)){j
                ,2}, PF(1).(PF_names(NotThere(k1,1),1)))

                ScoresE(1).(ScoresE_names(i,1))(1,j) =
                    ScoresE(1).(ScoresE_names(i,1))(1,j) - (
                    Wf(1).(Wf_names(NotThere(k1,1),1)));

                Scores_TrackE(1).(Scores_TrackE_names(i,1)){
                    k3+1,j} = PF(1).(PF_names(NotThere(k1,1)
                    ,1));

                k3 = k3+1;
            end
        end
    end
end

```



```

for j = 1:numN

    numSurgeries = size(NovicesS.(NSeqNames(j,1)),1);
    Variables = {};

    for i = 1:numSurgeries

        my_variables1 = sprintf('S%dP1', i);
        Variables(1,i) = {my_variables1};

    end

    DeductionReason = {};
    colcountT = 1;
    for i = 1:size(Scores_TrackN.(Scores_TrackN_names(
        j,1)),1)

        DeductionReason(1,1) = {'LevDistIntDeduction'
            };

        if (size(Scores_TrackN.(Scores_TrackN_names(j
            ,1)),1) > 1) && (i > 1)

            DeductionReason(1,i) = {sprintf('
                DeductionReason%d',i-1)};

        end

    end

    Variables = string(Variables);

    T_ScoreN.(T_ScoreN_names(j,1)) = array2table(
        ScoresN.(ScoresN_names(j,1)), 'VariableNames',
        Variables, 'RowNames', {sprintf('ScoresN%d:',j)
        });

    T_TrackN.(T_TrackN_names(j,1)) = array2table(
        Scores_TrackN.(Scores_TrackN_names(j,1)), '
        VariableNames', Variables, 'RowNames',
        DeductionReason);

end

```

```

%Experts:
for j = 1:numE

    numSurgeries = size(ExpertsS.(ESeqNames(j,1)),1);
    Variables = {};

    for i = 1:numSurgeries

        my_variables1 = sprintf('S%dP1', i);
        Variables(1,i) = {my_variables1};

    end

    DeductionReason = {};
    colcountT = 1;
    for i = 1:size(Scores_TrackE.(Scores_TrackE_names(
        j,1)),1)

        DeductionReason(1,1) = {'LevDistIntDeduction'
            };

        if (size(Scores_TrackE.(Scores_TrackE_names(j
            ,1)),1) > 1) && (i > 1)

            DeductionReason(1,i) = {sprintf('
                DeductionReason%d',i-1)};

        end

    end

end

Variables = string(Variables);

T_ScoreE.(T_ScoreE_names(j,1)) = array2table(
    ScoresE.(ScoresE_names(j,1)), 'VariableNames',
    Variables, 'RowNames', {sprintf('ScoresE%d:',j)
    });

T_TrackE.(T_TrackE_names(j,1)) = array2table(
    Scores_TrackE.(Scores_TrackE_names(j,1)), '
    VariableNames', Variables, 'RowNames',
    DeductionReason);

```

```
end
```

Code B.6: Matlab script for two partitioned surgery-score-calculation with user defined crucial feedback steps and weighting factors (second score)

```
%% Score_calculation with two partitioned surgery:

%since the crucial performance feedback steps differ
    for each surgery the
%user needs to provide some information about his
    surgery-steps he defined
%to be crucial for the score calculation.
disp(seqPF{1,1});

%First save the number of steps provided by the user
    's PF data-files
prompt = 'How many crucial feedback steps/
    definitions does your PF1 data-file have?(INPUT:
    Number of rows in Crucial_Feedback_Cath_Steps1-
    file; ENTER)\n';

numPF1 = input(prompt);

disp(seqPF{2,1});

prompt = 'How many crucial feedback steps/
    definitions does your PF2 data-file have?(INPUT:
    Number of rows in Crucial_Feedback_Cath_Steps2-
    file; ENTER)\n';

numPF2 = input(prompt);

%generate the variables needed for the score
    calculation with the crucial
%surgery steps:
    for j =1:numPF1

        %to understand what happens here take a look at
            the comment in
        %score_calc1.m
        my_variablesPF1 = sprintf('PF1_%d', j);
```

```

    PF1(1).(my_variablesPF1) = string(seqPF{1,1}(j,2)
    );
    PF1_names(j,1) = convertCharsToStrings(
        my_variablesPF1);

    my_variablesWf1 = sprintf('Wf1_%d', j);
    Wf1(1).(my_variablesWf1) = [];
    Wf1_names(j,1) = convertCharsToStrings(
        my_variablesWf1);

end

for j =1:numPF2

    my_variablesPF2 = sprintf('PF2_%d', j);
    PF2(1).(my_variablesPF2) = string(seqPF{2,1}(j,2)
    );
    PF2_names(j,1) = convertCharsToStrings(
        my_variablesPF2);

    my_variablesWf2 = sprintf('Wf2_%d', j);
    Wf2(1).(my_variablesWf2) = [];
    Wf2_names(j,1) = convertCharsToStrings(
        my_variablesWf2);

end

%First PF1:
disp(seqPF{1,1});
%There are three possible cases the program can
    distinguish from:
% - does a step appear?
% - does a step not appear?
% - how many times does a step appear?
%to define how the user wants the program to treat
    the provided steps, he
%is ask to define the possibilities for each step:
prompt = 'From your PF1-steps how many shall the
    programm treat as: if they do NOT APPEAR execute
    a deduction in the score?(INPUT: Number of pf-
    steps; ENTER)\n';
numNotThere1 = input(prompt);

```

```

if numNotThere1 ~= 0

disp('Type the row number of the specific step, then
hit Enter and do so for each step:(INPUT:Row
number; ENTER)');
for i = 1:numNotThere1

prompt = 'Row number of step: ';
%NotThere1 stores the row numbers of the steps the
programm shall make a
%deduction in the score if the step does not appear
in a sequence
NotThere1(i,1) = input(prompt);

end

end

numThere1 = numPF1 - numNotThere1;
if numNotThere1 ~= 0

disp('Type the specific row number of the steps you
want the program to treat as: if they DO APPEAR
execute a deduction in the score.(INPUT:Row
number; ENTER)');
for i = 1:numThere1

prompt = 'Row number of step: ';
%There1 stores the row numbers of the steps the
programm shall make a
%deduction in the score if the step does appear in a
sequence
There1(i,1) = input(prompt);

end

end

%with NumWdif1 the user can define a weighting for
each step which he
%wishes to be different from the minimal deduction
-0.5.
prompt = 'From your PF1-steps how many do you wish

```



```

    to weight stronger than -0.5?(INPUT:Number; ENTER
    )\n';
NumWdif1 = input(prompt);

%number of steps which have the minimal deduction
    factor assigned:
NumWIn1 = numPF1 - NumWdif1;

disp('First type the row number of the specific step
    . Secondly the weighting factor of the specific
    step. ');
disp('Do so for each step!(INPUT:Row number; ENTER;
    factor for weighting; ENTER)');

%WIn is the minimal deduction factor
WIn = 0.5;

for i = 1:NumWdif1

    prompt = 'Row number of step: ';
    %RowWdif1 stores the rows of the steps the user
        defines a stronger
    %weighting factor for in the first column and
        respectively the weighting factor
    %in the second column
    RowWdif1(i,1) = input(prompt);

    prompt = 'Weighting factor of step: ';
    RowWdif1(i,2) = input(prompt);
    Wf1(1).(Wf1_names((RowWdif1(i,1)),1)) = RowWdif1(i
        ,2)*WIn;

end

prompt = 'Of how many PF1-steps would you like to
    know the number of appearances?(INPUT:Number;
    ENTER)\n';
NumApp1 = input(prompt);
disp('Type the row number of these steps
    respectively.(INPUT:Row number; ENTER)');

for i = 1:NumApp1

```

```

prompt = 'Row number of step: ';
%RowApp1 stores the row numbers of the steps for
    which the user wishes the program to
%get the number of appearances
RowApp1(i,1) = input(prompt);

%the user can then define after which count of the
    step/steps he wants the
%program to undertake a deduction for each further
    count
prompt = 'After which count do you want the program
    to deduct -0.5 per further count?(INPUT:Number;
    ENTER)\n';
CountStart1(i,1) = input(prompt);

end

%As a last input the user needs to provide the row
    numbers of the steps he
%wants to assign a minimal weighting factor of -0.5
disp('From your PF1-steps which ones do you wish to
    weight with -0.5?(INPUT:Row number; ENTER)');
for i = 1:NumWIn1

    prompt = 'Row number of step: ';
    RowWIn1(i,1) = input(prompt);
    Wf1(1).(Wf1_names((RowWIn1(i,1)),1)) = WIn;

end

%Second PF2:

%The same user inputs as described above are aksed
    from the user and saved
%for the second PF2 file

disp(seqPF{2,1});
prompt = 'From your PF2-steps how many shall the
    programm treat as: if they do NOT APPEAR execute
    a deduction in the score?(INPUT:Number of pf-
    steps; ENTER)\n';
numNotThere2 = input(prompt);

```

```
if numNotThere2 ~= 0

disp('Type the row number of the specific step, then
hit enter and do so for each step:(INPUT:Row
number; ENTER)');
for i = 1:numNotThere2

prompt = 'Row number of step: ';
NotThere2(i,1) = input(prompt);

end

end

numThere2 = numPF2 - numNotThere2;
if numNotThere2 ~= 0

disp('Type the specific row number of the steps you
want the program to treat as: if they DO APPEAR
execute a deduction in the score.(INPUT:Row
number; ENTER)');
for i = 1:numThere2

prompt = 'Row number of step: ';
There2(i,1) = input(prompt);

end

end

prompt = 'From your PF2-steps how many do you wish
to weight stronger then -0.5?(INPUT:Number; ENTER
)\n';
NumWdif2 = input(prompt);
NumWIn2 = numPF2 - NumWdif2;

disp('First type the row number of the specific step
. Secondly the weighting factor of the specific
step. ');
disp('Do so for each step!(INPUT:Row number; ENTER;
factor for weighting; ENTER)');
WIn = 0.5;
```

```

for i = 1:NumWdif2

prompt = 'Row number of step: ';
RowWdif2(i,1) = input(prompt);

prompt = 'Weighting factor of step: ';
RowWdif2(i,2) = input(prompt);
Wf2(1).(Wf2_names((RowWdif2(i,1)),1)) = RowWdif2(i
    ,2)*WIn;

end

prompt = 'Of how many PF2-steps would you like to
    know the number of appearances?(INPUT:Number;
    ENTER)\n';
NumApp2 = input(prompt);
disp('Type the row number of these steps
    respectively.(INPUT:Row number; ENTER)');

for i = 1:NumApp2

prompt = 'Row number of step: ';
RowApp2(i,1) = input(prompt);

prompt = 'After which count do you want the program
    to deduct -0.5 per further count?(INPUT:Number;
    ENTER)\n';
CountStart2(i,1) = input(prompt);

end

disp('From your PF2-steps which ones do you wish to
    weight with -0.5?(INPUT:Row number; ENTER)');
for i = 1:NumWIn2

prompt = 'Row number of step: ';
RowWIn2(i,1) = input(prompt);
Wf2(1).(Wf2_names((RowWIn2(i,1)),1)) = WIn;

end

%calculate score with this information:
%Novices:

```

```

for i = 1:numN

    k4 = 1;

    for j = 1:size(NovicesS(1).(NSeqNames(i,1)),1)
%Part1:
        if numNotThere1 == 0

            k3 = 1;

            for k1 = 1:numPF1

                for k2 = 1:NumApp1

                    if ~strcmp(PF1(1).(PF1_names(k1,1)), PF1(1).(
                        PF1_names(RowApp1(k2,1),1)))

                        if contains(NovicesS(1).(NSeqNames(i,1)){j
                            ,2}, PF1(1).(PF1_names(k1,1)))

                            ScoresN(1).(ScoresN_names(i,1))(1,k4) =
                                ScoresN(1).(ScoresN_names(i,1))(1,k4) - (
                                    Wf1(1).(Wf1_names(k1,1)));

                            %Scores_TrackN(1).(Scores_TrackN_names(i,1))
                                {k3+1,k4} = PF1(1).(PF1_names(k1,1));
                            Scores_TrackN(1).(Scores_TrackN_names(i,1)){
                                k3+1,k4} = seqPF{1,1}{k1,1};

                            k3 = k3+1;

                        end

                    elseif strcmp(PF1(1).(PF1_names(k1,1)), PF1(1)
                        .(PF1_names(RowApp1(k2,1),1)))

                        if contains(NovicesS(1).(NSeqNames(i,1)){j
                            ,2}, PF1(1).(PF1_names(k1,1)))

                            count = length(strfind(string(NovicesS(1).(
                                NSeqNames(i,1)){j,2}), PF1(1).(PF1_names(

```

```

        k1,1))));
    if count > CountStart1(k2,1)
    CountMult1 = count - CountStart1(k2,1);
    ScoresN(1).(ScoresN_names(i,1))(1,k4) =
        ScoresN(1).(ScoresN_names(i,1))(1,k4) - (
            Wf1(1).(Wf1_names(k1,1))*CountMult1);

    %Scores_TrackN(1).(Scores_TrackN_names(i,1))
    {k3+1,k4} = PF1(1).(PF1_names(k1,1));
    Scores_TrackN(1).(Scores_TrackN_names(i,1)){
        k3+1,k4} = seqPF{1,1}{k1,1};

    k3 = k3+1;

    end

end

end

end

end

end

end

%Part2:
if numNotThere2 == 0

k3 = 1;

for k1 = 1:numPF2

    for k2 = 1:NumApp2

        if ~strcmp(PF2(1).(PF2_names(k1,1)), PF2(1).(
            PF2_names(RowApp2(k2,1),1)))

            if contains(NovicesS(1).(NSeqNames(i,1)){j
                ,3}, PF2(1).(PF2_names(k1,1)))

                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =

```

```

        ScoresN(1).(ScoresN_names(i,1))(1,k4+1) -
            (Wf2(1).(Wf2_names(k1,1)));

        %Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {k3+1,k4+1} = PF2(1).(PF2_names(k1,1));
        Scores_TrackN(1).(Scores_TrackN_names(i,1)){
            k3+1,k4+1} = seqPF{2,1}{k1,1};

        k3 = k3+1;

    end

elseif strcmp(PF2(1).(PF2_names(k1,1)), PF2(1)
    .(PF2_names(RowApp2(k2,1),1)))

    if contains(NovicesS(1).(NSeqNames(i,1)){j
        ,3}, PF2(1).(PF2_names(k1,1)))

        count = length(strfind(string(NovicesS(1).(
            NSeqNames(i,1)){j,3}), PF2(1).(PF2_names(
                k1,1))));
        if count > CountStart2(k2,1)
            CountMult2 = count - CountStart2(k2,1);
            ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) -
                    (Wf2(1).(Wf2_names(k1,1))*CountMult2);

            %Scores_TrackN(1).(Scores_TrackN_names(i,1))
                {k3+1,k4+1} = PF2(1).(PF2_names(k1,1));
            Scores_TrackN(1).(Scores_TrackN_names(i,1)){
                k3+1,k4+1} = seqPF{2,1}{k1,1};

            k3 = k3+1;

        end

    end

end

end

```

```

    end

end

if numNotThere1 >= 1

    k3 = 1;

    %Part1:
    for k1 = 1:numThere1

        for k2 = 1:NumApp1

            if ~strcmp(PF1(1).(PF1_names(There1(k1,1),1)),
                PF1(1).(PF1_names(RowApp1(k2,1),1)))

                if contains(NovicesS(1).(NSeqNames(i,1)){j
                    ,2}, PF1(1).(PF1_names(There1(k1,1),1)))

                    ScoresN(1).(ScoresN_names(i,1))(1,k4) =
                        ScoresN(1).(ScoresN_names(i,1))(1,k4) -
                        (Wf1(1).(Wf1_names(There1(k1,1),1)));

                    %Scores_TrackN(1).(Scores_TrackN_names(i,1)
                        ){k3+1,k4} = PF1(1).(PF1_names(There1(k1
                        ,1),1));
                    Scores_TrackN(1).(Scores_TrackN_names(i,1))
                        {k3+1,k4} = seqPF{1,1}{There1(k1,1),1};

                    k3 = k3+1;

                end

            elseif strcmp(PF1(1).(PF1_names(There1(k1,1)
                ,1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

                if contains(NovicesS(1).(NSeqNames(i,1)){j
                    ,2}, PF1(1).(PF1_names(There1(k1,1),1)))

                    count = length(strfind(string(NovicesS(1).(
                        NSeqNames(i,1)){j,2}), PF1(1).(PF1_names
                        (There1(k1,1),1))));
                    if count > CountStart1(k2,1)

```



```

        CountMult1 = count - CountStart1(k2,1);
        ScoresN(1).(ScoresN_names(i,1))(1,k4) =
            ScoresN(1).(ScoresN_names(i,1))(1,k4) -
            (Wf1(1).(Wf1_names(There1(k1,1),1))*
            CountMult1);

        %Scores_TrackN(1).(Scores_TrackN_names(i,1)
            ){k3+1,k4} = PF1(1).(PF1_names(There1(k1
            ,1),1));
        Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {k3+1,k4} = seqPF{1,1}{There1(k1,1),1};

        k3 = k3+1;

    end

end

end

end

end

for k1 = 1:numNotThere1

    for k2 = 1:NumApp1

        if ~strcmp(PF1(1).(PF1_names(NotThere1(k1,1),1)
            ), PF1(1).(PF1_names(RowApp1(k2,1),1)))

            if ~contains(NovicesS(1).(NSeqNames(i,1)){j
                ,2}, PF1(1).(PF1_names(NotThere1(k1,1),1)))

                ScoresN(1).(ScoresN_names(i,1))(1,k4) =
                    ScoresN(1).(ScoresN_names(i,1))(1,k4) - (
                    Wf1(1).(Wf1_names(NotThere1(k1,1),1)));

                %Scores_TrackN(1).(Scores_TrackN_names(i,1))
                    {k3+1,k4} = PF1(1).(PF1_names(NotThere1(
                    k1,1),1));
                Scores_TrackN(1).(Scores_TrackN_names(i,1)){

```

```

        k3+1,k4} = seqPF{1,1}{NotThere1(k1,1),1};

    k3 = k3+1;

end

elseif strcmp(PF1(1).(PF1_names(NotThere1(k1,1),1)), PF1(1).(PF1_names(RowApp1(k2,1),1)))

    if ~contains(NovicesS(1).(NSeqNames(i,1)){j,2}, PF1(1).(PF1_names(NotThere1(k1,1),1)))

        count = length(strfind(string(NovicesS(1).(NSeqNames(i,1)){j,2}), PF1(1).(PF1_names(NotThere1(k1,1),1))));
        if count > CountStart1(k2,1)
            CountMult1 = count - CountStart1(k2,1);
            ScoresN(1).(ScoresN_names(i,1))(1,k4) =
                ScoresN(1).(ScoresN_names(i,1))(1,k4) - (
                    Wf1(1).(Wf1_names(NotThere1(k1,1),1))*
                    CountMult1);

            %Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {k3+1,k4} = PF1(1).(PF1_names(NotThere1(k1,1),1));
            Scores_TrackN(1).(Scores_TrackN_names(i,1))
            {k3+1,k4} = seqPF{1,1}{NotThere1(k1,1),1};

        k3 = k3+1;

    end

end

end

end

end

end

```

```

if numNotThere2 >= 1

k3 = 1;

%Part2:
for k1 = 1:numThere2

    for k2 = 1:NumApp2

        if ~strcmp(PF2(1).(PF2_names(There2(k1,1),1))
            , PF2(1).(PF2_names(RowApp2(k2,1),1)))

            if contains(NovicesS(1).(NSeqNames(i,1)){j
                ,3}, PF2(1).(PF2_names(There2(k1,1),1)))

                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
                    ScoresN(1).(ScoresN_names(i,1))(1,k4+1)
                    - (Wf2(1).(Wf2_names(There2(k1,1),1)));

                %Scores_TrackN(1).(Scores_TrackN_names(i,1)
                    ){k3+1,k4+1} = PF2(1).(PF2_names(There2(
                        k1,1),1));
                Scores_TrackN(1).(Scores_TrackN_names(i,1)
                    ){k3+1,k4+1} = seqPF{2,1}{There2(k1,1)
                        ,1};

                k3 = k3+1;

            end

        elseif strcmp(PF2(1).(PF2_names(There2(k1,1)
            ,1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

            if contains(NovicesS(1).(NSeqNames(i,1)){j
                ,3}, PF2(1).(PF2_names(There2(k1,1),1)))

                count = length(strfind(string(NovicesS(1).(
                    NSeqNames(i,1)){j,3}), PF2(1).(PF2_names
                        (There2(k1,1),1))));
                if count > CountStart2(k2,1)
                    CountMult2 = count - CountStart2(k2,1);
                    ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =

```

```

        ScoresN(1).(ScoresN_names(i,1))(1,k4+1)
        - (Wf2(1).(Wf2_names(There2(k1,1),1))*
        CountMult2);

    %Scores_TrackN(1).(Scores_TrackN_names(i,1)
    ){k3+1,k4+1} = PF2(1).(PF2_names(There2(
    k1,1),1));
    Scores_TrackN(1).(Scores_TrackN_names(i,1))
    {k3+1,k4+1} = seqPF{2,1}{There2(k1,1)
    ,1};

    k3 = k3+1;

    end

end

end

end

end

for k1 = 1:numNotThere2

    for k2 = 1:NumApp2

        if ~strcmp(PF2(1).(PF2_names(NotThere2(k1,1),1)
        ), PF2(1).(PF2_names(RowApp2(k2,1),1)))

            if ~contains(NovicesS(1).(NSeqNames(i,1)){j
            ,3}, PF2(1).(PF2_names(NotThere2(k1,1),1)))

                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) =
                ScoresN(1).(ScoresN_names(i,1))(1,k4+1) -
                (Wf2(1).(Wf2_names(NotThere2(k1,1),1)));

                %Scores_TrackN(1).(Scores_TrackN_names(i,1)
                ){k3+1,k4+1} = PF2(1).(PF2_names(NotThere2
                (k1,1),1));
                Scores_TrackN(1).(Scores_TrackN_names(i,1))
                {k3+1,k4+1} = seqPF{2,1}{NotThere2(k1,1)

```

```
,1}];

k3 = k3+1;

end

elseif strcmp(PF2(1).(PF2_names(NotThere2(k1,1),1)), PF2(1).(PF2_names(RowApp2(k2,1),1)))

if ~contains(NovicesS(1).(NSeqNames(i,1)){j,3}, PF2(1).(PF2_names(NotThere2(k1,1),1)))

count = length(strfind(string(NovicesS(1).(NSeqNames(i,1)){j,3}), PF2(1).(PF2_names(NotThere2(k1,1),1))));
if count > CountStart2(k2,1)
CountMult2 = count - CountStart2(k2,1);
ScoresN(1).(ScoresN_names(i,1))(1,k4+1) = ScoresN(1).(ScoresN_names(i,1))(1,k4+1) - (Wf2(1).(Wf2_names(NotThere2(k1,1),1))* CountMult2);

%Scores_TrackN(1).(Scores_TrackN_names(i,1)){k3+1,k4+1} = PF2(1).(PF2_names(NotThere2(k1,1),1));
Scores_TrackN(1).(Scores_TrackN_names(i,1)){k3+1,k4+1} = seqPF{2,1}{NotThere2(k1,1),1};

k3 = k3+1;

end

end

end

end

end
```

```

    k4 = k4+2;

end

end

%Experts:
for i = 1:numE

    k4 = 1;

    for j = 1:size(ExpertsS(1).(ESeqNames(i,1)),1)

        if numNotThere1 == 0

            k3 = 1;

            %Part1:
            for k1 = 1:numPF1

                for k2 = 1:NumApp1

                    if ~strcmp(PF1(1).(PF1_names(k1,1)), PF1(1).(
                        PF1_names(RowApp1(k2,1),1)))

                        if contains(ExpertsS(1).(ESeqNames(i,1)){j
                            ,2}, PF1(1).(PF1_names(k1,1)))

                            ScoresE(1).(ScoresE_names(i,1))(1,k4) =
                                ScoresE(1).(ScoresE_names(i,1))(1,k4) - (
                                    Wf1(1).(Wf1_names(k1,1)));

                            %Scores_TrackE(1).(Scores_TrackE_names(i,1))
                                {k3+1,k4} = PF1(1).(PF1_names(k1,1));
                            Scores_TrackE(1).(Scores_TrackE_names(i,1)){
                                k3+1,k4} = seqPF{1,1}{k1,1};

                            k3 = k3+1;

                        end
                    end
                end
            end
        end
    end
end

```

```

elseif strcmp(PF1(1).(PF1_names(k1,1)), PF1(1)
    .(PF1_names(RowApp1(k2,1),1)))

    if contains(ExpertsS(1).(ESeqNames(i,1)){j
        ,2}, PF1(1).(PF1_names(k1,1)))

        count = length(strfind(string(ExpertsS(1).(
            ESeqNames(i,1)){j,2}), PF1(1).(PF1_names(
                k1,1))));
        if count > CountStart1(k2,1)
            CountMult1 = count - CountStart1(k2,1);
            ScoresE(1).(ScoresE_names(i,1))(1,k4) =
                ScoresE(1).(ScoresE_names(i,1))(1,k4) - (
                    Wf1(1).(Wf1_names(k1,1))*CountMult1);

            %Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {k3+1,k4} = PF1(1).(PF1_names(k1,1));
            Scores_TrackE(1).(Scores_TrackE_names(i,1)){
                k3+1,k4} = seqPF{1,1}{k1,1};

            k3 = k3+1;

        end

    end

end

end

end

end

end

end

%Part2:
if numNotThere2 == 0

    k3 = 1;

    for k1 = 1:numPF2

        for k2 = 1:NumApp2

```

```

if ~strcmp(PF2(1).(PF2_names(k1,1)), PF2(1).(
    PF2_names(RowApp2(k2,1),1)))

    if contains(ExpertsS(1).(ESeqNames(i,1)){j
        ,3}, PF2(1).(PF2_names(k1,1)))

        ScoresE(1).(ScoresE_names(i,1))(1,k4+1) =
            ScoresE(1).(ScoresE_names(i,1))(1,k4+1) -
            (Wf2(1).(Wf2_names(k1,1)));

        %Scores_TrackE(1).(Scores_TrackE_names(i,1))
            {k3+1,k4+1} = PF2(1).(PF2_names(k1,1));
        Scores_TrackE(1).(Scores_TrackE_names(i,1)){
            k3+1,k4+1} = seqPF{2,1}{k1,1};

        k3 = k3+1;

    end

elseif strcmp(PF2(1).(PF2_names(k1,1)), PF2(1)
    .(PF2_names(RowApp2(k2,1),1)))

    if contains(ExpertsS(1).(ESeqNames(i,1)){j
        ,3}, PF2(1).(PF2_names(k1,1)))

        count = length(strfind(string(ExpertsS(1).(
            ESeqNames(i,1)){j,3}), PF2(1).(PF2_names(
                k1,1))));
        if count > CountStart2(k2,1)
            CountMult2 = count - CountStart2(k2,1);
            ScoresE(1).(ScoresE_names(i,1))(1,k4+1) =
                ScoresE(1).(ScoresE_names(i,1))(1,k4+1) -
                (Wf2(1).(Wf2_names(k1,1))*CountMult2);

            %Scores_TrackE(1).(Scores_TrackE_names(i,1))
                {k3+1,k4+1} = PF2(1).(PF2_names(k1,1));
            Scores_TrackE(1).(Scores_TrackE_names(i,1)){
                k3+1,k4+1} = seqPF{2,1}{k1,1};

            k3 = k3+1;

        end
    end

```



```

        end

    end

end

end

end

```

Code B.7: Matlab function for data import with automatic adaption to single or two partitioned surgeries

```

%% Import data from text file

function S2 = importfile2(filename,UserInput)

%% Setup the Import Options

% prompt = 'Does the surgery you wish to analyse
%          have two characteristic parts as defined by the
%          seperation of left leg (LL) and right leg (RL)
%          seen in the example Novice1.txt or not?(2 or 1)\n
%          ';
%
% UserInput = input(prompt);

if UserInput == 2

    Var = 3;
    opts.VariableNames = ["Participants", "Part1", "
        Part2"];
    opts.VariableTypes = ["string", "string", "
        string"];

elseif UserInput == 1

    Var = 2;
    opts.VariableNames = ["Participants", "Sequences
        "];

```

```

        opts.VariableTypes = ["string", "string"];

else disp('WRONG INPUT; START AGAIN');

end

opts = delimitedTextImportOptions("NumVariables",
    Var);

% Specify range and delimiter
opts.DataLines = [2, Inf];
opts.Delimiter = "\t";

% Specify column names and types
% opts.VariableNames = ["Novices", "SequencesLL", "
    SequencesRL"];
% opts.VariableTypes = ["string", "string", "string
    "];
opts = setvaropts(opts, [1, 2], "WhitespaceRule", "
    preserve");
opts = setvaropts(opts, [1, 2], "EmptyFieldRule", "
    auto");
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
S2 = readtable(filename, opts);

%% Convert to output type
S2 = table2cell(S2);
numIdx = cellfun(@(x) ~isnan(str2double(x)), S2);
S2(numIdx) = cellfun(@(x) {str2double(x)}, S2(numIdx
    ));

end

```

Code B.8: Matlab function for user provided crucial performance feedback steps import

```

%% Import data from text file

function S4 = importfile4(filename)

```

```

%% Setup the Import Options
opts = delimitedTextImportOptions("NumVariables", 2)
    ;

% Specify range and delimiter
opts.DataLines = [2, Inf];
opts.Delimiter = "\t";

% Specify column names and types
opts.VariableNames = ["SurgerySteps", "Abbreviations
    "];
opts.VariableTypes = ["string", "string"];
opts = setvaropts(opts, [1, 2], "WhitespaceRule", "
    preserve");
opts = setvaropts(opts, [1, 2], "EmptyFieldRule", "
    auto");
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
S4 = readtable(filename, opts);

%% Convert to output type
S4 = table2cell(S4);
numIdx = cellfun(@(x) ~isnan(str2double(x)), S4);
S4(numIdx) = cellfun(@(x) {str2double(x)}, S4(numIdx
    ));

end

```

## B.2 AOI Sequence Analysis Code

All the functions used in the following scripts are attached at the end.

Code B.9: Matlab script for AOI sequences analysis from eye tracking data

```

clear all;
clc;

%% Add path to data-folder

```

```
addpath('P:\04_Student_theses\1378
_BereiterLivio_BA_TSP_AutomatedProcessAnalysis\
Program\sequence_analysis\eye_track_seq\Data');

%% import sequences from all participants and
   experts and store them in a cell array

numN = 1;

numE = 1;

prompt = 'Does your data-set have two characteristic
   parts like Part1 and Part2 for each trial?(INPUT
   :2 or 1 respectively)\n';

UserParts = input(prompt);

prompt = 'How many trials does your experiment have
   ?(INPUT:Number)\n';

if UserParts == 2

    numTrials = input(prompt);
    Variables(1,1) = {'Participants'};
    colcount = 1;

    for i = 1:numTrials

        my_variables1 = sprintf('T%dP1', i);
        Variables11(1,i) = {my_variables1};

    end

    for i = 1:numTrials

        my_variables2 = sprintf('T%dP2', i);
        Variables22(1,i) = {my_variables2};

    end

    for i = 1:2:(2*numTrials)
```

```

        Variables(1,i+1) = Variables11(1,colcount);
        Variables(1,i+2) = Variables22(1,colcount);

        colcount = colcount + 1;

    end

    Variables = string(Variables);
    numTrials = (2*numTrials) + 1;

elseif UserParts == 1

    numTrials = input(prompt);
    Variables(1,1) = {'Participants'};

    for i = 1:numTrials

        my_variables1 = sprintf('T%dP1', i);
        Variables(1,i+1) = {my_variables1};

    end

    Variables = string(Variables);
    numTrials = numTrials + 1;

end

%import novice-data:
DataN(1,1) = {importfile3('Participants_Type1.txt',
    Variables, numTrials)};

%import expert-data:
DataE(1,1) = {importfile3('Experts_Type1.txt',
    Variables, numTrials)};

%import template:

disp('Does your Template_Typei.txt-file contain a
    single sequence per part of the experiment');
disp('or does it contain a whole set ranging for
    example from Trial1 to Trial9?(INPUT:Single or
    Whole)');

```

```

prompt='';

LO_SH = input(prompt, 's');

if strcmp(LO_SH, 'Single')

    if UserParts == 2

        numTrials = 2+1;

        VariablesT = ["Participants", "P1", "P2"];

    elseif UserParts == 1

        numTrials = 1+1;

        VariablesT = ["Participants", "P1"];

    end

elseif strcmp(LO_SH, 'Whole')

    VariablesT = Variables;

end

DataT(1,1) = {importfile3('Template_Type3.txt',
    VariablesT, numTrials)};

% import Find_Type1.txt file for customized
% analysisation

DataF(1,1) = {importfile4('Find_Type1.txt')};

%disp(DataF{1,1});

%% shorten sequences to not successive characters;
% example AAABBCCCCABBC to ABCABC
while (true)

    while (true)
        prompt = 'Do you want to shorten the sequences? For
            example from AAABBCCCCABBC to ABCABC?(INPUT:

```

```

        yes or no)\n';

Gonogo = input(prompt, 's');

if strcmp(Gonogo, 'yes')

%Participants:
    for i1 = 1:size(DataN,1)

        for i2 = 1:size(DataN{i1,1},1)

            SeqShort_N{i1,1}{i2,1} = convertStringsToChars(
                DataN{i1,1}{i2,1});

            for j = 2:size(DataN{i1,1},2)

                if length(convertStringsToChars(DataN{i1,1}{i2,j})) < 2

                    SeqShort_N{i1,1}{i2,j} = [];

                else

                    SeqShort_N{i1,1}{i2,j} = seq_slimmer(
                        convertStringsToChars(DataN{i1,1}{i2,j}));

                end

            end

        end

    end

end

%Experts:
    for i1 = 1:size(DataE,1)

        for i2 = 1:size(DataE{i1,1},1)

            SeqShort_E{i1,1}{i2,1} = convertStringsToChars(
                DataE{i1,1}{i2,1});


```

```

        for j = 2:size(DataE{i1,1},2)

            if length(convertStringsToChars(DataE{i1,1}{i2,j})) < 2

                SeqShort_E{i1,1}{i2,j} = 'too short';

            else

                SeqShort_E{i1,1}{i2,j} = seq_slimmer(
                    convertStringsToChars(DataE{i1,1}{i2,j}));

            end

        end

    end

end

%Templates:

    for i1 = 1:size(DataT,1)

    for i2 = 1:size(DataT{i1,1},1)

        SeqShort_T{i1,1}{i2,1} = convertStringsToChars(
            DataT{i1,1}{i2,1});

        for j = 2:size(DataT{i1,1},2)

            if length(convertStringsToChars(DataT{i1,1}{i2,j})) < 2

                SeqShort_T{i1,1}{i2,j} = 'too short';

            else

                SeqShort_T{i1,1}{i2,j} = seq_slimmer(
                    convertStringsToChars(DataT{i1,1}{i2,j}));

            end

```



```

        end

    end

end

end

break;

elseif strcmp(Gonogo, 'no')

    disp('The sequences are not shortened');
    SeqShort_N{1,1} = cellstr(DataN{1,1});
    SeqShort_E{1,1} = cellstr(DataE{1,1});
    SeqShort_T{1,1} = cellstr(DataT{1,1});
    break;

else

    disp('Wrong INPUT, start again');

end

end

break;

end
%% Calculate Levenshtein dist and Damerau-
Levenshtein dist between Expert and Participants
Variables = cell(convertStringsToChars(Variables));

[Distances,UserDec,AppearanceN,AppearanceE,
 T_A_N_names,T_A_E_names] = score_calc2(SeqShort_N
 , SeqShort_E, SeqShort_T, DataF, UserParts, numN,
 numE, Variables, LO_SH);

if strcmp(UserDec, 'no')

disp('

```

---

```

    ');
disp('The sequences of the novices and experts were
      compared with the beforehand provided template');
disp('and these were the corresponding Levenshtein
      distances:');
disp(Distances.DistN);
disp(Distances.DistE);

elseif strcmp(UserDec, 'yes')

disp('
-----
    ');
disp('The sequences of the novices and experts were
      compared with the beforehand provided template');
disp('and these were the corresponding Levenshtein
      distances(top table) and Damerau-Levenshtein
      distances(bottom table):');
disp(Distances.DistN{1,1});
disp(Distances.DistN{2,1});
disp(Distances.DistE{1,1});
disp(Distances.DistE{2,1});

end

% writetable(Distances.DistN, '
    Participants_Type2_LevD_to_TiPi_short.xlsx');
% writetable(Distances.DistE, '
    Experts_Type2_LevD_to_TiPi_short.xlsx');

%% find user-defined AOI's or patterns of AOI's

disp('
-----
    ');
disp('The sequences have been analysed according to
      the custom inputs and were stored in the
      Find_Type1_P/E_Appearances.xlsx:');
disp('Each sheet stands for the corresponding
      Participant/Expert provided in your text-files.')
;

% for i = 1:size(SeqShort_N{1,1},1)

```

```

%
%   writetable(AppearanceN.(T_A_N_names(i,1)), '
%   Find_Type1_Participants_Appearances.xlsx', 'Sheet
%   ', i);
%
% end
%
% for i = 1:size(SeqShort_E{1,1},1)
%
%   writetable(AppearanceE.(T_A_E_names(i,1)), '
%   Find_Type1_Experts_Appearances.xlsx', 'Sheet', i)
%   ;
%
% end

```

Code B.10: Matlab script to calculate edit distances with Levenshtein or Damerau-Levenshtein distance algorithm

```

%%function for eye-tracking perf. feedback:

function [Distances,UserDistance,AppearanceN,
    AppearanceE,T_A_N_names,T_A_E_names] =
    score_calc2(seqN, seqE, seqT, seqF, UserInput,
    numN, numE, VariableNames, All_Trials)
%% Let user choose template if there is one provided
:

NRow = 1;
ERow = 1;

if (UserInput == 2) && strcmp(All_Trials, 'Single')

    TEMPL_P1 = string(seqT{1,1}{1,2});
    TEMPL_P2 = string(seqT{1,1}{1,3});

elseif (UserInput == 1) && strcmp(All_Trials, '
    Single')

    TEMPL_P1 = string(seqT{1,1}{1,2});

elseif (UserInput == 2) && strcmp(All_Trials, '
    Whole')

```

```

    countcol1 = 1;

    for i = 2:2:size(seqT{1,1},2)

        TEMPL_P1(1,countcol1) = string(seqT{1,1}{1,i});
        TEMPL_P2(1,countcol1) = string(seqT{1,1}{1,i+1});

        countcol1 = countcol1+1;
    end

elseif (UserInput == 1) && strcmp(All_Trials, '
    Whole')

    countcol1 = 1;

    for i = 2:size(seqT{1,1},2)

        TEMPL_P1(1,countcol1) = string(seqT{1,1}{1,i});

        countcol1 = countcol1+1;
    end

end

%% assign data to number of participant-variables(
    Novices/Experts) to calculate edit-distances:

for j = 1:numN

    my_variablesN = sprintf('seq_N%d', j);
    NovicesS(1).(my_variablesN) = seqN{j,1};
    NSeqNames(j,1) = convertCharsToStrings(
        my_variablesN);

    my_variablesN = sprintf('L_P1_N%d', j);
    NovicesD1(1).(my_variablesN) = {};
    N_NamesD1(j,1) = convertCharsToStrings(
        my_variablesN);

    my_variablesN = sprintf('D_L_P1_N%d', j);
    NovicesD1(1).(my_variablesN) = {};
    N_NamesD1(j+numN,1) = convertCharsToStrings(

```

```

        my_variablesN);

my_variablesN = sprintf('L_P2_N%d', j);
NovicesD2(1).(my_variablesN) = {};
N_NamesD2(j,1) = convertCharsToStrings(
    my_variablesN);

my_variablesN = sprintf('D_L_P2_N%d', j);
NovicesD2(1).(my_variablesN) = {};
N_NamesD2(j+numN,1) = convertCharsToStrings(
    my_variablesN);

my_variablesN = sprintf('allD_N%d', j);
AllDN(1).(my_variablesN) = [];
AllDN_names(j,1) = convertCharsToStrings(
    my_variablesN);

my_variablesN = sprintf('TN%d', j);
my_variablesN_L = sprintf('L_DN%d', j);
my_variablesN_DL = sprintf('DL_DN%d', j);
AllTN(1).(my_variablesN) = [];
AllTN_names(j,1) = convertCharsToStrings(
    my_variablesN);
AllTN_names(j,2) = convertCharsToStrings(
    my_variablesN_L);
AllTN_names(j,3) = convertCharsToStrings(
    my_variablesN_DL);
%
%     my_variablesN = sprintf('ScoresN%d', j);
%     ScoresN(1).(my_variablesN) = [];
%     ScoresN_names(j,1) = convertCharsToStrings(
%         my_variablesN);

end

for j = 1:size(seqN{1,1},1)

    my_variablesN = sprintf('ScoresN%d', j);
    ScoresN(1).(my_variablesN) = [];
    ScoresN_names(j,1) = convertCharsToStrings(
        my_variablesN);

    my_variablesN = sprintf('T_A_N%d', j);

```

```

T_A_N(1).(my_variablesN) = [];
T_A_N_names(j,1) = convertCharsToStrings(
    my_variablesN);

end

for j = 1:numE

    my_variablesE = sprintf('seq_E%d', j);
    ExpertsS(1).(my_variablesE) = seqE{j,1};
    ESeqNames(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('L_P1_E%d', j);
    ExpertsD1(1).(my_variablesE) = {};
    E_NamesD1(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('D_L_P1_E%d', j);
    ExpertsD1(1).(my_variablesE) = {};
    E_NamesD1(j+numE,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('L_P2_E%d', j);
    ExpertsD2(1).(my_variablesE) = {};
    E_NamesD2(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('D_L_P2_E%d', j);
    ExpertsD2(1).(my_variablesE) = {};
    E_NamesD2(j+numE,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('allD_E%d', j);
    AllDE(1).(my_variablesE) = [];
    AllDE_names(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('TE%d', j);
    my_variablesE_L = sprintf('L_DE%d', j);
    my_variablesE_DL = sprintf('DL_DE%d', j);
    AllTE(1).(my_variablesE) = [];
    AllTE_names(j,1) = convertCharsToStrings(

```

```

        my_variablesE);
    AllTE_names(j,2) = convertCharsToStrings(
        my_variablesE_L);
    AllTE_names(j,3) = convertCharsToStrings(
        my_variablesE_DL);

%     my_variablesE = sprintf('ScoresE%d', j);
%     ScoresE(1).(my_variablesE) = [];
%     ScoresE_names(j,1) = convertCharsToStrings(
%         my_variablesE);

end

for j = 1:size(seqE{1,1},1)

    my_variablesE = sprintf('ScoresE%d', j);
    ScoresE(1).(my_variablesE) = [];
    ScoresE_names(j,1) = convertCharsToStrings(
        my_variablesE);

    my_variablesE = sprintf('T_A_E%d', j);
    T_A_E(1).(my_variablesE) = [];
    T_A_E_names(j,1) = convertCharsToStrings(
        my_variablesE);

end
%% calculate distances for single partitioned
surgery

if UserInput == 1

while (true)

while (true)

    prompt = 'Would you like to calculate the
        Levenshtein-Dist. and the Damerau-Levenshtein-
        Dist.?(INPUT:yes or no)\n';
    UserDistance = input(prompt, 's');

    if strcmp(UserDistance, 'no')
%%Levenshtein-dist.:

```

```

%Novices:

for j = 1:size(NovicesS(1).(NSeqNames(NRow,1)),1)

    colcount = 1;

    for k = 2:size(NovicesS(1).(NSeqNames(NRow,1)),2)

        NovicesD1(1).(N_NamesD1(NRow,1)){j,k-1} =
            L_distance(char(convertStringsToChars(
                TEMPL_P1(1,colcount))),char(
                convertStringsToChars(NovicesS(1).(
                    NSeqNames(NRow,1)){j,k})));

        if strcmp(All_Trials, 'Whole')
            colcount = colcount + 1;
        end
    end

end

AllDN(1).(AllDN_names(NRow,1)){1,1} =
    cell2mat(NovicesD1(1).(N_NamesD1(NRow,1))
    );

AllTN(1).(AllTN_names(NRow,1)) = array2table
    (AllDN(1).(AllDN_names(NRow,1)){1,1}, '
    VariableNames', VariableNames(1,2:end), '
    RowNames', ...
    NovicesS(1).(NSeqNames(NRow,1))(1:end,1));

%Experts:

for j = 1:size(ExpertsS(1).(ESeqNames(numE,1)),1)

    colcount = 1;
    for k = 2:size(ExpertsS(1).(ESeqNames(ERow,1)),2)

        ExpertsD1(1).(E_NamesD1(ERow,1)){j,k-1} =

```



```

        L_distance(char(convertStringsToChars(
            TEMPL_P1(1,colcount))),char(
            convertStringsToChars(ExpertsS(1).(
                ESeqNames(ERow,1)){j,k})));
    if strcmp(All_Trials, 'Whole')
        colcount = colcount + 1;
    end
end

end

AllDE(1).(AllDE_names(ERow,1)){1,1} =
    cell2mat(ExpertsD1(1).(E_NamesD1(ERow,1))
);

AllTE(1).(AllTE_names(ERow,1)) = array2table
    (AllDE(1).(AllDE_names(ERow,1)){1,1}, '
    VariableNames', VariableNames(1,2:end), '
    RowNames', ...
    ExpertsS(1).(ESeqNames(ERow,1))(1:end,1));

Distances(1).DistN = AllTN(1).(AllTN_names(
    NRow,1));
Distances(1).DistE = AllTE(1).(AllTE_names(
    ERow,1));
break;

%% calculate Levenshtein-Dist. and Damerau-
Levenshtein-Dist.

%choose sequences in seq array to calculate edit
distance in comparison to
%template
elseif strcmp(UserDistance, 'yes')

%Levenshtein Dist.:
%Novices
    for j = 1:size(NovicesS(1).(NSeqNames(NRow,1))
        ,1)

        colcount = 1;
        for k = 2:size(NovicesS(1).(NSeqNames(NRow
            ,1)),2)

```

```

        NovicesD1(1).(N_NamesD1(NRow,1)){j,k-1} =
            L_distance(char(convertStringsToChars(
                TEMPL_P1(1,colcount))),char(
                convertStringsToChars(NovicesS(1).(
                    NSeqNames(NRow,1)){j,k})));
        if strcmp(All_Trials, 'Whole')
            colcount = colcount + 1;
        end
    end

end

%Experts:
for j = 1:size(ExpertsS(1).(ESeqNames(numE,1)),1)

    colcount = 1;
    for k = 2:size(ExpertsS(1).(ESeqNames(ERow,1)),2)

        ExpertsD1(1).(E_NamesD1(ERow,1)){j,k-1} =
            L_distance(char(convertStringsToChars(
                TEMPL_P1(1,colcount))),char(
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(ERow,1)){j,k})));
        if strcmp(All_Trials, 'Whole')
            colcount = colcount + 1;
        end
    end

end

%Damerau-Levenshtein Dist.:
%Novices:
for j = 1:size(NovicesS(1).(NSeqNames(NRow,1)),1)

    colcount = 1;
    for k = 2:size(NovicesS(1).(NSeqNames(NRow,1)),2)

        NovicesD2(1).(N_NamesD2(NRow,1)){j,k-1} =

```

```

        damerau_levenshtein(char(
            convertStringsToChars(TEMPL_P1(1,
                colcount))),char(convertStringsToChars(
                NovicesS(1).(NSeqNames(NRow,1)){j,k})));
    if strcmp(All_Trials, 'Whole')
        colcount = colcount + 1;
    end
end

end

AllDN(1).(AllDN_names(NRow,1)){1,1} = cell2mat(
    NovicesD1(1).(N_NamesD1(NRow,1)));
AllDN(1).(AllDN_names(NRow,1)){2,1} = cell2mat(
    NovicesD2(1).(N_NamesD2(NRow,1)));

AllTN(1).(AllTN_names(NRow,1)){1,1} =
    array2table(AllDN(1).(AllDN_names(NRow,1))
        {1,1}, 'VariableNames', VariableNames(1,2:end
        ), ...
        'RowNames', NovicesS(1).(NSeqNames(NRow,1))
            (1:end,1));
AllTN(1).(AllTN_names(NRow,1)){2,1} =
    array2table(AllDN(1).(AllDN_names(NRow,1))
        {2,1}, 'VariableNames', VariableNames(1,2:end
        ), ...
        'RowNames', NovicesS(1).(NSeqNames(NRow
            ,1))(1:end,1));

%Experts:
for j = 1:size(ExpertsS(1).(ESeqNames(numE,1))
    ,1)

    colcount = 1;
    for k = 2:size(ExpertsS(1).(ESeqNames(ERow
        ,1)),2)

        ExpertsD2(1).(E_NamesD2(ERow,1)){j,k-1} =
            damerau_levenshtein(char(
                convertStringsToChars(TEMPL_P1(1,
                    colcount))),char(convertStringsToChars(
                    ExpertsS(1).(ESeqNames(ERow,1)){j,k})));
        if strcmp(All_Trials, 'Whole')

```

```

        colcount = colcount + 1;
    end
end

end

AllDE(1).(AllDE_names(ERow,1)){1,1} =
    cell2mat(ExpertsD1(1).(E_NamesD1(ERow,1))
    );
AllDE(1).(AllDE_names(ERow,1)){2,1} =
    cell2mat(ExpertsD2(1).(E_NamesD2(ERow,1))
    );

AllTE(1).(AllTE_names(ERow,1)){1,1} =
    array2table(AllDE(1).(AllDE_names(ERow,1)
    ){1,1}, 'VariableNames', VariableNames
    (1,2:end), ...
    'RowNames', ExpertsS(1).(ESeqNames(ERow
    ,1))(1:end,1));
AllTE(1).(AllTE_names(ERow,1)){2,1} =
    array2table(AllDE(1).(AllDE_names(ERow,1)
    ){2,1}, 'VariableNames', VariableNames
    (1,2:end), ...
    'RowNames', ExpertsS(1).(ESeqNames(ERow
    ,1))(1:end,1));

Distances(1).DistN = AllTN(1).(AllTN_names(
    NRow,1));
Distances(1).DistE = AllTE(1).(AllTE_names(
    ERow,1));

break;

else disp('WRONG INPUT');

end

end

break;

end

```

```

%% calculate distances for two partitioned surgery

elseif UserInput == 2

while (true)

while (true)

    prompt = 'Would you like to calculate the
              Levenshtein-Dist. and the Damerau-Levenshtein-
              Dist.?(INPUT:yes or no)\n';
    UserDistance = input(prompt, 's');

    if strcmp(UserDistance, 'no')
%%Levenshtein-dist.:

%Novices:

        for j = 1:size(NovicesS(1).(NSeqNames(NRow,1))
            ,1)

            colcount = 1;

            for k = 2:2:size(NovicesS(1).(NSeqNames(NRow
                ,1)),2)

                NovicesD1(1).(N_NamesD1(NRow,1)){j,k-1} =
                    L_distance(char(convertStringsToChars(
                        TEMPL_P1(1,colcount))),char(
                        convertStringsToChars(NovicesS(1).(
                            NSeqNames(NRow,1)){j,k})));

                NovicesD1(1).(N_NamesD1(NRow,1)){j,k} =
                    L_distance(char(convertStringsToChars(
                        TEMPL_P2(1,colcount))),char(
                        convertStringsToChars(NovicesS(1).(
                            NSeqNames(NRow,1)){j,k+1})));

                if strcmp(All_Trials, 'Whole')
                    colcount = colcount + 1;
                end
            end
        end
    end
end

```

```

end

AllDN(1).(AllDN_names(NRow,1)){1,1} =
    cell2mat(NovicesD1(1).(N_NamesD1(NRow,1))
    );

AllTN(1).(AllTN_names(NRow,1)) = array2table
    (AllDN(1).(AllDN_names(NRow,1)){1,1}, '
    VariableNames', VariableNames(1,2:end), '
    RowNames', ...
    NovicesS(1).(NSeqNames(NRow,1))(1:end,1));

%Experts:

for j = 1:size(ExpertsS(1).(ESeqNames(numE,1))
,1)

    colcount = 1;

    for k = 2:2:size(ExpertsS(1).(ESeqNames(ERow
,1)),2)

        ExpertsD1(1).(E_NamesD1(ERow,1)){j,k-1} =
            L_distance(char(convertStringsToChars(
                TEMPL_P1(1,colcount))),char(
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(ERow,1)){j,k})));

        ExpertsD1(1).(E_NamesD1(ERow,1)){j,k} =
            L_distance(char(convertStringsToChars(
                TEMPL_P2(1,colcount))),char(
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(ERow,1)){j,k+1})));

        if strcmp(All_Trials, 'Whole')
            colcount = colcount + 1;
        end
    end
end

end

AllDE(1).(AllDE_names(ERow,1)){1,1} =
    cell2mat(ExpertsD1(1).(E_NamesD1(ERow,1))

```

```

    );

    AllTE(1).(AllTE_names(ERow,1)) = array2table
        (AllDE(1).(AllDE_names(ERow,1)){1,1}, '
        VariableNames', VariableNames(1,2:end), '
        RowNames', ...
        ExpertsS(1).(ESeqNames(ERow,1))(1:end,1));

    Distances(1).DistN = AllTN(1).(AllTN_names(
        NRow,1));
    Distances(1).DistE = AllTE(1).(AllTE_names(
        ERow,1));

    break;

%% calculate Levenshtein-Dist. and Damerau-
    Levenshtein-Dist.

%choose sequences in seq array to calculate edit
    distance in comparison to
%template
    elseif strcmp(UserDistance,'yes')

%Levenshtein Dist.:
%Novices
    for j = 1:size(NovicesS(1).(NSeqNames(NRow,1))
        ,1)

        colcount = 1;

        for k = 2:2:size(NovicesS(1).(NSeqNames(NRow
            ,1)),2)

            NovicesD1(1).(N_NamesD1(NRow,1)){j,k-1} =
                L_distance(char(convertStringsToChars(
                    TEMPL_P1(1,colcount))),char(
                    convertStringsToChars(NovicesS(1).(
                        NSeqNames(NRow,1)){j,k})));

            NovicesD1(1).(N_NamesD1(NRow,1)){j,k} =
                L_distance(char(convertStringsToChars(
                    TEMPL_P2(1,colcount))),char(

```

```

        convertStringsToChars(NovicesS(1).(
            NSeqNames(NRow,1)){j,k+1})));

    if strcmp(All_Trials, 'Whole')
        colcount = colcount + 1;
    end
end

end

%Experts:
for j = 1:size(ExpertsS(1).(ESeqNames(numE,1))
    ,1)

    colcount = 1;

    for k = 2:2:size(ExpertsS(1).(ESeqNames(ERow
        ,1)),2)

        ExpertsD1(1).(E_NamesD1(ERow,1)){j,k-1} =
            L_distance(char(convertStringsToChars(
                TEMPL_P1(1,colcount))),char(
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(ERow,1)){j,k})));

        ExpertsD1(1).(E_NamesD1(ERow,1)){j,k} =
            L_distance(char(convertStringsToChars(
                TEMPL_P2(1,colcount))),char(
                convertStringsToChars(ExpertsS(1).(
                    ESeqNames(ERow,1)){j,k+1})));

        if strcmp(All_Trials, 'Whole')
            colcount = colcount + 1;
        end
    end

end

end

%Damerau-Levenshtein Dist.:
%Novices:

for j = 1:size(NovicesS(1).(NSeqNames(NRow,1))

```



```

,1)

colcount = 1;

for k = 2:2:size(NovicesS(1).(NSeqNames(NRow
,1)),2)

    NovicesD2(1).(N_NamesD2(NRow,1)){j,k-1} =
        damerau_levenshtein(char(
            convertStringsToChars(TEMPL_P1(1,
            colcount))),char(convertStringsToChars(
            NovicesS(1).(NSeqNames(NRow,1)){j,k})));

    NovicesD2(1).(N_NamesD2(NRow,1)){j,k} =
        damerau_levenshtein(char(
            convertStringsToChars(TEMPL_P2(1,
            colcount))),char(convertStringsToChars(
            NovicesS(1).(NSeqNames(NRow,1)){j,k+1})))
        );

    if strcmp(All_Trials, 'Whole')
        colcount = colcount + 1;
    end

end

end

AllDN(1).(AllDN_names(NRow,1)){1,1} =
    cell2mat(NovicesD1(1).(N_NamesD1(NRow,1))
    );
AllDN(1).(AllDN_names(NRow,1)){2,1} =
    cell2mat(NovicesD2(1).(N_NamesD2(NRow,1))
    );

AllTN(1).(AllTN_names(NRow,1)){1,1} =
    array2table(AllDN(1).(AllDN_names(NRow,1)
    ){1,1}, 'VariableNames', VariableNames
    (1,2:end), ...
    'RowNames', NovicesS(1).(NSeqNames(NRow
    ,1))(1:end,1));
AllTN(1).(AllTN_names(NRow,1)){2,1} =
    array2table(AllDN(1).(AllDN_names(NRow,1)

```

```

        ){2,1}, 'VariableNames', VariableNames
        (1,2:end), ...
        'RowNames', NovicesS(1).(NSeqNames(NRow
        ,1))(1:end,1));

%Experts:

for j = 1:size(ExpertsS(1).(ESeqNames(ERow,1))
,1)

    colcount = 1;

    for k = 2:2:size(ExpertsS(1).(ESeqNames(ERow
    ,1)),2)

        ExpertsD2(1).(E_NamesD2(ERow,1)){j,k-1} =
            damerau_levenshtein(char(
                convertStringsToChars(TEMPL_P1(1,
                colcount))),char(convertStringsToChars(
                ExpertsS(1).(ESeqNames(ERow,1)){j,k})));

        ExpertsD2(1).(E_NamesD2(ERow,1)){j,k} =
            damerau_levenshtein(char(
                convertStringsToChars(TEMPL_P2(1,
                colcount))),char(convertStringsToChars(
                ExpertsS(1).(ESeqNames(ERow,1)){j,k+1})))
            );

        if strcmp(All_Trials, 'Whole')
            colcount = colcount + 1;
        end

    end

end

AllDE(1).(AllDE_names(ERow,1)){1,1} =
    cell2mat(ExpertsD1(1).(E_NamesD1(ERow,1))
    );
AllDE(1).(AllDE_names(ERow,1)){2,1} =
    cell2mat(ExpertsD2(1).(E_NamesD2(ERow,1))
    );

```

```

        AllTE(1).(AllTE_names(ERow,1)){1,1} =
            array2table(AllDE(1).(AllDE_names(ERow,1)
            ){1,1}, 'VariableNames', VariableNames
            (1,2:end), ...
            'RowNames', ExpertsS(1).(ESeqNames(ERow
            ,1))(1:end,1));
        AllTE(1).(AllTE_names(ERow,1)){2,1} =
            array2table(AllDE(1).(AllDE_names(ERow,1)
            ){2,1}, 'VariableNames', VariableNames
            (1,2:end), ...
            'RowNames', ExpertsS(1).(ESeqNames(ERow
            ,1))(1:end,1));

        Distances(1).DistN = AllTN(1).(AllTN_names(
            NRow,1));
        Distances(1).DistE = AllTE(1).(AllTE_names(
            ERow,1));

        break;

    else disp('WRONG INPUT');

    end

end

    break;

end

end

%% Find specific AOI/AOI-pattern appearances in
    sequences:

score_calc2_custom;

end

```

Code B.11: Matlab script to find specific AOIs or AOI-patterns within the sequences and count their appearances

```

%% Score_calculation with single partitioned surgery
:
disp('
-----
');
disp(seqF{1,1});

prompt = 'How many AOI-patterns or single AOIs does
your Find_Typei.txt file contain?(INPUT:number)\n
';

numPF = input(prompt);

for j =1:numPF

    my_variablesPF = sprintf('PF_%d', j);
    PF(1).(my_variablesPF) = string(seqF{1,1}(j,2));
    PF_names(j,1) = convertCharsToStrings(
        my_variablesPF);

end

prompt = 'Of how many PF-steps would you like to
know the number of appearances?(INPUT:number)\n';
NumApp = input(prompt);
disp('Now type the row number of these steps
respectively.(INPUT:Number of row)');

for i = 1:NumApp

prompt = 'Row number: ';
RowApp(i,1) = input(prompt);

end

for j = 1:size(NovicesS(1).(NSeqNames(NRow,1)),1)

    for k = 2:size(NovicesS(1).(NSeqNames(NRow,1)),2)

        k2 = 1;

```

```

for k1 = 1:numPF

    if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
        PF_names(RowApp(k2,1),1)))

        if contains(NovicesS(1).(NSeqNames(NRow,1)){j
            ,k}, PF(1).(PF_names(k1,1)))

            ScoresN(1).(ScoresN_names(j,1)){k1,k} = 1;

        else ScoresN(1).(ScoresN_names(j,1)){k1,k} =
            0;

        end

    elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
        PF_names(RowApp(k2,1),1)))

        if contains(NovicesS(1).(NSeqNames(NRow,1)){j
            ,k}, PF(1).(PF_names(k1,1)))

            count = length(strfind(string(NovicesS(1).(
                NSeqNames(NRow,1)){j,k}), PF(1).(PF_names
                    (k1,1))));
            ScoresN(1).(ScoresN_names(j,1)){k1,k} =
                count;

        else ScoresN(1).(ScoresN_names(j,1)){k1,k} =
            0;

        end

        k2 = k2+1;

    end

end

end

for i = 1:size(seqF{1,1},1)

```

```

        ScoresN(1).(ScoresN_names(j,1)){i,1} = seqF
            {1,1}{i,2};

    end

end

%Experts:

for j = 1:size(ExpertsS(1).(ESeqNames(ERow,1)),1)

    for k = 2:size(ExpertsS(1).(ESeqNames(ERow,1)),2)

        k2 = 1;

        for k1 = 1:numPF

            if ~strcmp(PF(1).(PF_names(k1,1)), PF(1).(
                PF_names(RowApp(k2,1),1)))

                if contains(ExpertsS(1).(ESeqNames(ERow,1)){j
                    ,k}, PF(1).(PF_names(k1,1)))

                    ScoresE(1).(ScoresE_names(j,1)){k1,k} = 1;

                else ScoresE(1).(ScoresE_names(j,1)){k1,k} =
                    0;

                end

                elseif strcmp(PF(1).(PF_names(k1,1)), PF(1).(
                    PF_names(RowApp(k2,1),1)))

                    if contains(ExpertsS(1).(ESeqNames(ERow,1)){j
                        ,k}, PF(1).(PF_names(k1,1)))

                        count = length(strfind(string(ExpertsS(1).(
                            ESeqNames(ERow,1)){j,k}), PF(1).(PF_names
                                (k1,1))));
                        ScoresE(1).(ScoresE_names(j,1)){k1,k} =
                            count;

                    else ScoresE(1).(ScoresE_names(j,1)){k1,k} =

```

```

        0;

    end

    k2 = k2+1;

end

end

end

for i = 1:size(seqF{1,1},1)

    ScoresE(1).(ScoresE_names(j,1)){i,1} = seqF
        {1,1}{i,2};

end

end

%% Store Appearances in tables for each Participant/
Expert:

VariableNames{1,1} = 'AOIsToFind';

for i = 1:size(T_A_N_names,1)

    T_A_N(1).(T_A_N_names(i,1)) = array2table(
        ScoresN(1).(ScoresN_names(i,1)), '
        VariableNames', VariableNames(1,1:end));

end

for i = 1:size(T_A_E_names,1)

    T_A_E(1).(T_A_E_names(i,1)) = array2table(
        ScoresE(1).(ScoresE_names(i,1)), '
        VariableNames', VariableNames(1,1:end));

end

AppearanceN = T_A_N;
AppearanceE = T_A_E;

```

---

Code B.12: Matlab function to shorten sequences if desired by the user

```
%% function to shorten Sequences:

function C1 = seq_slimmer(sequence)

%walk through sequence and compare if momentarily
    char1 is different from
%following char2 and if so save char1 to SeqShort
for k = 2:length(sequence)

    CharChosen = sequence(k-1);

    NextChar = sequence(k);

    if CharChosen ~= NextChar

        SeqShort(k-1) = CharChosen;

    end

end

%the last char-type of sequence is lost if it is
    different from the one before
%it due to the beforehand defined walk through.
    Therefore the last char
%must be compared singlehandedly with the one before
    it and if different
%added at the end of SeqShort
if sequence(length(sequence)) ~= sequence(length(
sequence)-1)

    SeqShort(length(SeqShort)+1) = sequence(length(
sequence));

elseif sequence(length(sequence)) == sequence(length(
sequence)-1)

    SeqShort(length(SeqShort)+1) = sequence(length(
sequence));
```



```

end

%assign output as the new sequence without any
spaces inbetween --> as a single "word"
C1 = cellstr(SeqShort(find(~isspace(SeqShort)))));

end

```

Code B.13: Matlab function to import participant data with dynamic generation of columns according to number of trials in the experiment

```

%% Import data from text file

function S3 = importfile3(filename, VariableNames,
    numCol)

%% Setup the Import Options
opts = delimitedTextImportOptions("NumVariables",
    numCol);

for i = 1:numCol

    VariableTypes(1,i) = ["string"];

end

% Specify range and delimiter
opts.DataLines = [2, Inf];
opts.Delimiter = "\t";

% Specify column names and types
opts.VariableNames = VariableNames;
opts.VariableTypes = VariableTypes;
opts = setvaropts(opts, [1, 2], "WhitespaceRule", "
    preserve");
opts = setvaropts(opts, [1, 2], "EmptyFieldRule", "
    auto");
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data

```

```

S3 = readtable(filename, opts);

%% Convert to output type
S3 = table2cell(S3);
numIdx = cellfun(@(x) ~isnan(str2double(x)), S3);
S3(numIdx) = cellfun(@(x) {str2double(x)}, S3(numIdx
    ));

end

```

The importfile4.m function is the same as the one used in [B.1](#).

## B.3 Functions/Algorithms for Sequence Analysis

Code B.14: Levenshtein distance algorithm

```

%% function for levenshtein distance

function [D,d] = L_distance(str1,str2)

m = length(str1);
n = length(str2);

d = zeros(n+1,m+1);

for i=1:n
    d(i+1,1)=i;
end

for j=1:m
    d(1,j+1)=j;
end

for i=1:n
    for j=1:m
        if (str2(i) == str1(j))
            d(i+1,j+1)=d(i,j);
        else

```

```

            d(i+1,j+1)=1+min(min(d(i+1,j),d(i,j+1)),
                               d(i,j));
        end
    end
end

D=d(n+1,m+1);

end

```

Code B.15: Damerau-Levenshtein distance algorithm

```

%% function for Damerau-Levenshtein distance

function [D,d] = damerau_levenshtein(str1,str2)

%save length of strings to compare

m = length(str1);
n = length(str2); %template

%% set up matrix

% two extra rows and collums to save high cost at
% first row/column
% and minimum distance for each character at second
% row/column

m2 = m+2;
n2 = n+2;

d = zeros(n2,m2);

%define high cost higher than maximum distance
% possible between both strings,
%it helps to simplify logic of the algorithm;

%for further explanation see below

max_dist = m + n;

for i=1:n2

```

```
        d(i,1) = max_dist;
end

for j=1:n

    d(j+2,2) = j;
end

for k=1:m2

    d(1,k) = max_dist;
end

for l=1:m

    d(2,l+2) = l;
end

%% set up array for 'row remembering' of specific
    character from str1 at ASCII as index
% 1 stays for no match yet, which would lead
    directly to row or column 1
% --> max_dist value

last_row_ch_str2 = ones(1,256);

% for f = 1:256
%
%     last_row_ch_str1(f) = 1;
%
% end

%% calculate distances insertion/deletion/
    replacement/transposition

% i is the running variable for the row
for i = 1:n

    %assign ASCII of i-th character from str1
```

```

ch_str2 = double(str2(i));

%stores the column where the j-th character of
    str2 was seen last
%set last_match_col as well to 1 --> no match
    yet
last_match_col = 1;

% j is the running variable for the column
for j = 1:m

    %assign ASCII of j-th character from str2
    ch_str1 = double(str1(j));

    %assign row of last appearance of j-th
        character
    last_matching_row = last_row_ch_str2(ch_str1
        );

    %matching characters?
    if (ch_str2 == ch_str1)

        cost = 0;

    else cost = 1;

    end

    dist_ins = d(i+1,j+2) + 1; %insertion of
        character
    dist_del = d(i+2,j+1) + 1; %deletion of
        character
    dist_rep = d(i+1,j+1) + cost; %replacement
        of character

    %% Transposition

    %Transposition cost is: (cost before
        transposition) + (distance
        between rows) + (distance between columns)
        + 1

    %once the algorithm realices with

```

```
        last_matching_row and
%last_match_col if there was a matching char
    like str2(j) in a row before and
%in which column the last general match was
    respectively, it
%automatically finds a low 'cost before
    transposition' since
%last_matching_row and last_match_col are
    larger than 1.

%--> get away from first row/column with
    max_dist values

%because all char between the transposition
    are treated like
%insertions or deletions, and
    damerau_levenshtein only allows for
%either one of them, one can check this with
    the following:

%i-last_matching_row: must be zero if no
    addition of a char is
%needed

%j-last_match_col: must be zero if no
    deletion of a char is
%needed

%--> if neither one of them is zero both
    would be required and
%adding together both will cause a total
    cost of a transposition
%much larger than other operations.

dist_transp = d(last_matching_row,
    last_match_col) + (i-last_matching_row) +
    (j-last_match_col) +1; %transposition of
    two characters

%% Assign minimum distance to current cell

%min_dist = min(min(min(dist_ins,dist_del),
    dist_rep),dist_transp);
```

```

        d(i+2,j+2) = min(min(min(dist_ins,dist_del),
                                dist_rep),dist_transp);

        %if there was a general match in the current
            row = i, the column needs to be
        %stored in last_match_col;

        %last_match_col and last_matching_row are 1
            lower than they should
        %be to simplify logic

        if (cost == 0)

            last_match_col = j+1;

        end

    end

    %when one steps a row = i further, the row of
        the current char from
    %str1(i) must be stored at its ASCII position
        since one wants to know
    %in which row what character was

        last_row_ch_str2(ch_str2) = i+1;

end

D = d(n2,m2);
end

```

Code B.16: Common Sub-Sequence function

```

function [substr,loc1,loc2] = commonsubstring(str1,
    substr,substringlength)
% commonsubstring - finds the longest common
    substring(s) between two strings
% usage: [substr,loc1,loc2] = commonsubstring(str1,
    str2)

```

```
% usage: [substr,loc1,loc2] = commonsubstring(str1,  
    str2,substringlength)  
%  
% Find the longest common substrings between a pair  
    of strings,  
% or if the substring length is indicated, finds all  
    common  
% substrings with that specified length.  
%  
% Arguments: (input)  
%   str1, str2 - character strings that will be  
    searched for  
%       common substrings.  
%  
%   substringlength - (OPTIONAL) - scalar positive  
    integer  
%       if provided. This specifies the length of  
    the common  
%       substrings to be generated. If not provided  
    or empty  
%       the maximum length common substrings are  
    returned.  
%  
%       Note, if str1 and str2 contain many  
    thousands of  
%       elements, then searching for a very long  
    substring  
%       of a specific length may be a time and  
    memory consuming  
%       operation.  
%  
% Arguments: (output)  
%   substr - character string that contains the  
    longest common  
%       substring. If more than one string has that  
    maximum  
%       length, then substr will be a matrix of  
    strings.  
%       If multiple solutions were found, then each  
    row of  
%       that character matrix will be a solution.  
%  
%   loc1, loc2 - the respective starting locations of
```



```

    these
%       substrings in each of the original strings
provided.
%       Since a given substring may be found
multiple times,
%       these will be cell arrays, possibly
containing vectors
%       of indices
%
% Example:
% % Generate a pair of long random letter sequences,
then
% % determine the longest common substring between
them
% bases = 'acgt';
% str1 = bases(ceil(rand(1,100000)*4));
% str2 = bases(ceil(rand(1,100000)*4));
%
% tic,[substr,ind1,ind2] = commonsubstring(str1,
str2);toc
% % Elapsed time is 16.650532 seconds.
%
% % There were two substrings of the maximum length
% % (16) characters.
% substr
% % substr =
% % gcttttagggcggtacgc
% % cttcggataccttggt
%
% ind1
% % ind1 =
% %      [22189]
% %      [74425]
%
% ind2
% % ind2 =
% %      [64948]
% %      [32833]
%
% Example:
% % Find all common substrings of a given fixed
length.
% str1 = char('a' + round(rand(1,100)*1.5))

```

```

% % str1 =
% %
    bbbabbbbbbabbabbbbabbbbabbbababbbabbbabbbbbbabbabbbbbbabbabbaaabb
% str2 = 'aaabbabbb';
%
% [substr,ind1,ind2] = commonsubstring(str1,str2,3)
% % substr =
% % aaa
% % aab
% % abb
% % bab
% % bba
% % bbb
%
% % ind1 =
% %      [1x2  double]
% %      [1x4  double]
% %      [1x15 double]
% %      [1x11 double]
% %      [1x15 double]
% %      [1x19 double]
%
% % ind2 =
% %      [          1]
% %      [          2]
% %      [1x2  double]
% %      [          5]
% %      [          4]
% %      [          7]
%
% See also: regexp, strfind, findstr
%
% Author: John D'Errico
% e-mail: woodchips@rochester.rr.com
% Release date: 4/28/2010

if (nargin < 2) || (nargin > 3)
    error('COMMONSUBSTRING:improperarguments', ...
        'Either 2 or 3 arguments must be supplied')
end

if (nargin < 3)

```

```
    substrlength = [];  
elseif ~isnumeric(substrlength) || (numel(  
    substrlength) > 1) || ...  
    isinf(substrlength) || isnan(substrlength)  
    || ...  
    (rem(substrlength,1) ~= 0) || (  
        substrlength < 1)  
  
    error('COMMONSUBSTRING:improperarguments', ...  
        'substrlength must be positive, scalar,  
        finite integer if provided')  
end  
  
if ~ischar(str1) || ~ischar(str2)  
    error('COMMONSUBSTRING:improperarguments', ...  
        'Both arguments must be strings')  
elseif isempty(str1) || isempty(str2)  
    % If either of these strings are empty, the  
    % common substring must be also  
    substr = '';  
    loc1 = {};  
    loc2 = {};  
    return  
end  
  
% unroll the strings into row vectors  
str1 = str1(:).';  
str2 = str2(:).';  
nstr1 = numel(str1);  
nstr2 = numel(str2);  
nstr = min(numel(str1),numel(str2));  
  
% was a specific substring length indicated?  
if ~isempty(substrlength)  
    % yes  
    if substrlength > nstr  
        substr = '';  
        loc1 = {};  
        loc2 = {};  
    end  
  
    % get the unique substrings of that length  
    if nstr1 <= nstr2
```

```
% str1 is the shorter of the two, or they
% have the same lengths. In either event,
% take the list of possible substrings
% from str1.
substr = substrings(str1,substringlength,1);
else
    % str2 is the shorter of the two
    substr = substrings(str2,substringlength,1);
end

% get the locations of each of these strings
cellsubstr = cellstr(substr);
loc1 = regexp(str1,cellsubstr);
loc2 = regexp(str2,cellsubstr);

% delete those substrings that were not found
% in both original strings.
k = cellfun(@isempty,loc1) | cellfun(@isempty,loc2);
substr(k,:) = [];
loc1(k) = [];
loc2(k) = [];

% we can exit now
return
end

% extract the single character string elements of
% str1
substr = unique(str1');
cellsubstr = cellstr(substr);

% Location of these single character substrings in
% str2
% regexp will give this quite efficiently
ind2 = regexp(str2,cellsubstr);

% drop those characters that are not located in str2
k2 = cellfun(@isempty,ind2);
substr(k2) = [];
ind2(k2) = [];
cellsubstr(k2) = [];
```

```
% Location of these single character substrings in
    str1
ind1 = regexp(str1,cellsubstr);

% were there any common single digit strings?
% if not, then we are done already
if isempty(substr)
    substr = '';
    loc1 = {};
    loc2 = {};
    return
end

% now, see if we can extend any of these substrings
% by just a single character at a time
flag = true;
substrlength = 1;
while flag
    nsubstr = size(substr,1);
    ext = cell(nsubstr,3);
    L = 0;
    for i = 1:nsubstr
        substr_i = substr(i,:);

        % Where did we find this string in each of
        % str1 and str2?
        loc1 = ind1{i};
        loc2 = ind2{i};

        % we can drop any occurrences that lie
        % at the very beginning of str1 or str2
        loc1(loc1 == 1) = [];
        loc2(loc2 == 1) = [];

        % find the very next characters that lie just
        % prior to this substring in str1 and str2.
        % pc refers to the previous character, to be
        % then prepended.
        pc1 = str1(loc1 - 1);
        pc2 = str2(loc2 - 1);

        % are there any common characters? If
        % so, then we can extend the corresponding
```

```

% substrings
pccommon = intersect(pc1,pc2);
ncommon = numel(pccommon);
if ncommon > 0
    extind1 = cell(ncommon,1);
    extind2 = cell(ncommon,1);
    L = L + 1;
    for j = 1:ncommon
        extind1{j} = loc1(pc1 == pccommon(j)) - 1;
        extind2{j} = loc2(pc2 == pccommon(j)) - 1;
    end

    ext{L,1} = [pccommon', repmat(substri,ncommon
        ,1)];
    ext{L,2} = extind1;
    ext{L,3} = extind2;
end % if ncommon > 0
end % for i = 1:nsubstr

% were we able to prepend any new characters to
% extend
% this list of substrings?
if (L >= 1) && (substrlength < nstr)
    % yes,so continue prepending
    substrlength = substrlength + 1;

    substr = cell2mat(ext(1:L,1));
    ind1 = cat(1,ext{1:L,2});
    ind2 = cat(1,ext{1:L,3});
else
    % no, so terminate with the previous results
    loc1 = ind1;
    loc2 = ind2;
    flag = false;
end

end % while flag

```

Code B.17: Sub-Strings function used in Common Sub-Sequence function

```
function strs = substrings(str,nsub,uniqueflag)
```

```

% substrings: extract all substrings of a given
%   length from a vector
% usage: strs = substrings(str, nsub)
% usage: strs = substrings(str, nsub, uniqueflag)
%
%
% arguments: (input)
%   str - any vector string, of any class.
%
%   nsub - scalar, positive integer - specifies the
%         length
%         of the substrings to be generated
%
%   uniqueflag - (OPTIONAL) scalar boolean flag that
%               specifies if the result will be allowed to
%               contain duplicate entries, or will only
%               those unique substrings be returned.
%
%               uniqueflag == 0 --> return EVERY substring
%               from str. This array will not be sorted
%               .
%               uniqueflag == 1 --> Only the unique (sorted
%               )
%               list of substrings will be generated.
%               This array will be sorted.
%
% arguments: (output)
%   strs - a array of sub-strings, one row for each
%         substring found in str.
%
%         Note: substrings(str,1) is the same as
%               unique(substrings(str), 'rows')
%
%
% Example:
%   bases = 'acgt';
%   str = bases(ceil(rand(1,20)*4))
%   str =
%   attcgcgtagcctagatgttt
%
%   % Find ALL substrings, replicates are allowed.
%   substrings(str,2)

```

```
% ans =
% at
% tt
% tc
% cg
% gc
% cg
% gt
% tg
% gc
% cc
% ct
% ta
% ag
% ga
% at
% tg
% gt
% tt
% tt
%
% % Find the set of distinct, unique substrings
% substrings(str,2,1)
% ans =
% ag
% at
% cc
% cg
% ct
% ga
% gc
% gt
% ta
% tc
% tg
% tt
%
% See also: strtok, cellstr, strfun, allwords
%
% Author: John D'Errico
% e-mail: woodchips@rochester.rr.com
% Release date: 4/7/2010
```



```
% test for problems.
% first, verify that str is a vector.
if ~isvector(str)
    error('SUBSTRINGS:nonvectorinput','str must be a
        vector')
end
if nargin > 3
    error('SUBSTRINGS:toomanyargs', ...
        'No more than two arguments allowed')
elseif nargin < 1
    help substrings
    return
end

if (nargin < 3) || isempty(uniqueflag)
    uniqueflag = 0;
elseif ~ismember(uniqueflag,[0 1])
    error('SUBSTRINGS:improperarg', ...
        'uniqueflag must be 0 or 1 if supplied.')
end

if nargin < 2
    error('SUBSTRINGS:insufficientargs', ...
        'At least two arguments must be supplied.')
end
if isempty(nsub) || ~isnumeric(nsub) || (nsub <= 0)
    || (nsub ~= round(nsub))
    error('SUBSTRINGS:improperarg', ...
        'nsub must be positive, integer, scalar, and
        numeric.')
end

% the number of elements in str
n = length(str);

% empty begets empty, but also if nsub is longer
% than n
% we must return empty.
if isempty(str) || (nsub > n)
    strs = [];
    return
elseif n == nsub
    % special case, with only one substring
```

```
    strs = str;
    return
elseif nsub == 1
    % special case, substrings of length 1
    strs = reshape(str,[],1);
    % do we get the unique elements?
    if uniqueflag
        strs = unique(strs);
    end
    return
end

% ensure that str is a column vector
str = str(:);

% create the indices of all substrings of
% the given length
ind = bsxfun(@plus,(0:(n - nsub))',1:nsub);
strs = str(ind);

% do we need to make them unique?
if uniqueflag
    strs = unique(strs,'rows');
end
```

## List of Figures

2.1	Intervention steps to successfully catheterize a vessel according to Seldinger technique(Ivar Seldinger (2008)). . . . .	5
2.2	Specific labelling of the femoral artery and vein catheterization steps performed previously of the MitraClip intervention. . . .	6
2.3	Example sequence of a surgery according to beforehand defined labelling; Middle: Left leg sequence; Right: Right leg sequence.	6
2.4	Template sequence for a correct and smooth vessel catheterization; Middle: Left leg sequence; Right: Right leg sequence. .	6
2.5	Dynamic matrix (light grey) generated by Levenshtein distance algorithm, where on the bottom right the minimum number of operations to change "benyam" to "ephrem", is presented; Arrows: indicate the optimal reversed path, where each operation is a replacement (diagonal direction). . . . .	8
2.6	Example for an expert's definition of the crucial task steps/-subsequences; US: Ultrasound. . . . .	10
2.7	Example for user inputs in order to analyse surgical sequences with specific task-step algorithm. . . . .	10
2.8	Example for a complete AOI sequence from a usability study; A: Building area; B: Building blocks; C: Computer(Gianduzzo (2020)). . . . .	13
2.9	Example for user provided AOIs or AOI-patterns to further analyse the sequences(Gianduzzo (2020)). . . . .	14
3.1	Architecture of the surgeon skill assessment program; PF: Performance feedback. . . . .	16

3.2	Visualization of the .txt file dependency of the program's performance feedback; N: Novice; E: Expert; T: Template; PF: Crucial performance feedback steps; S1P1: Part 1 of surgery 1; S1P2: Part 2 of surgery 1; S2P1: Part 1 of surgery 2; S2P2: Part 2 of surgery 2. . . . .	16
3.3	Crucial performance feedback steps/subsequences and weighting factors for the left leg catheterization. . . . .	17
3.4	Crucial performance feedback steps/subsequences and weighting factors for the right leg catheterization. . . . .	17
3.5	First half of the catheterization scores and deduction reasons for Novice1, Novice2, Novice3, Expert1, Expert2. . . . .	18
3.6	Second half of the catheterization scores and deduction reasons for Novice1, Novice2, Novice3, Expert1, Expert2. . . . .	19
3.7	Program architecture of the AOI sequence analysis. . . . .	20
A.1	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	27
A.2	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	28
A.3	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	29
A.4	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	30
A.5	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	31
A.6	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	32
A.7	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	33
A.8	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	34
A.9	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	35
A.10	Results of Celine Gianduzzo's bachelor thesis in terms of AOI sequence analysis(Gianduzzo (2020)). . . . .	36

A.11 Results of Celine Gianduzzo's bachelor thesis in terms of AOI	
sequence analysis(Gianduzzo (2020)). . . . .	37

# Codeverzeichnis

B.1 Header-Matlab script for sequence analysis . . . . .	38
B.2 Matlab script for surgical sequence analysis . . . . .	42
B.3 Matlab script for surgeon score calculation (first score) . . . .	45
B.4 Matlab script for catheterization-score-calculation with self de- fined crucial performance feedback steps and weighting factors (second score) . . . . .	71
B.5 Matlab script for single partitioned surgery-score-calculation with user defined crucial feedback steps and weighting factors (second score) . . . . .	89
B.6 Matlab script for two partitioned surgery-score-calculation with user defined crucial feedback steps and weighting factors (second score) . . . . .	103
B.7 Matlab function for data import with automatic adaption to single or two partitioned surgeries . . . . .	122
B.8 Matlab function for user provided crucial performance feedback steps import . . . . .	123
B.9 Matlab script for AOI sequences analysis from eye tracking data	124
B.10 Matlab script to calculate edit distances with Levenshtein or Damerau-Levenshtein distance algorithm . . . . .	132
B.11 Matlab script to find specific AOIs or AOI-patterns within the sequences and count their appearances . . . . .	149
B.12 Matlab function to shorten sequences if desired by the user . .	153
B.13 Matlab function to import participant data with dynamic generation of columns according to number of trials in the experiment . . . . .	154
B.14 Levenshtein distance algorithm . . . . .	155
B.15 Damerau-Levenshtein distance algorithm . . . . .	156
B.16 Common Sub-Sequence function . . . . .	160
B.17 Sub-Strings function used in Common Sub-Sequence function .	167

# Bibliography

- Ahmed, K., Miskovic, D., Darzi, A., Athanasiou, T. & Hanna, G. B. (2011), 'Observational tools for assessment of procedural skills: a systematic review', *The American Journal of Surgery* **202**(4), 469--480.
- Ahmed, N., Devitt, K. S., Keshet, I., Spicer, J., Imrie, K., Feldman, L., Cools-Lartigue, J., Kayssi, A., Lipsman, N., Elmi, M. et al. (2014), 'A systematic review of the effects of resident duty hour restrictions in surgery: impact on resident wellness, training, and patient outcomes', *Annals of surgery* **259**(6), 1041.
- Cumin, D. (2020), 'Longest common subsequence', <https://www.mathworks.com/matlabcentral/fileexchange/24559-longest-common-subsequence>, Zugriff: 03.01.2020.
- Gianduzzo, C. (2020), 'Gaze behavior dependency on task complexity: A game based approach to usability evaluations'.
- Harezlak, K. & Kasprowski, P. (2018), 'Application of eye tracking in medicine: A survey, research issues and challenges', *Computerized Medical Imaging and Graphics* **65**, 176--190.
- Ivar Seldinger, S. (2008), 'Catheter replacement of the needle in percutaneous arteriography: a new technique', *Acta radiologica* **49**(suppl\_434), 47--52.
- Khan, R. S., Tien, G., Atkins, M. S., Zheng, B., Panton, O. N. & Meneghetti, A. T. (2012), 'Analysis of eye gaze: Do novice surgeons look at the same location as expert surgeons during a laparoscopic operation?', *Surgical endoscopy* **26**(12), 3536--3540.
- Kok, E. M., Jarodzka, H., de Bruin, A. B., BinAmir, H. A., Robben, S. G. & van Merriënboer, J. J. (2016), 'Systematic viewing in radiology: seeing more, missing less?', *Advances in Health Sciences Education* **21**(1), 189--205.

- Larson, J. L., Williams, R. G., Ketchum, J., Boehler, M. L. & Dunnington, G. L. (2005), 'Feasibility, reliability and validity of an operative performance rating system for evaluating surgery residents', *Surgery* **138**(4), 640-649.
- Levenshtein, V. I. (1966), Binary codes capable of correcting deletions, insertions, and reversals, *in* 'Soviet physics doklady', Vol. 10, pp. 707--710.
- Richstone, L., Schwartz, M. J., Seideman, C., Cadeddu, J., Marshall, S. & Kavoussi, L. R. (2010), 'Eye metrics as an objective assessment of surgical skill', *Annals of surgery* **252**(1), 177--182.
- Ristad, E. S. & Yianilos, P. N. (1998), 'Learning string-edit distance', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(5), 522--532.
- Saner, H. (2014), 'e-cardiology and e-health: from industry-driven technical progress to clinical application', *Eur J Prev Cardiol* **21**(2 Suppl), 2--3.
- Su, Z., Ahn, B.-R., Eom, K.-Y., Kang, M.-K., Kim, J.-P. & Kim, M.-K. (2008), 'Plagiarism detection using the levenshtein distance and smith-waterman algorithm', pp. 569--569.