IMAGE ANALYSIS TECHNIQUES FOR VERTEBRA ANOMALY DETECTION IN X-RAY IMAGES

by

MOHAMMED DAS

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2008

Approved by

Dr. Fikret Ercal, Co - Advisor	Dr. R. Joe Stanley, Co - Advisor
Dr. Bruce M. McMillin	Dr. Randy H. Moss

ABSTRACT

In this research, imaging techniques are investigated for the analysis and detection of abnormalities in cervical and lumbar vertebrae. Detecting vertebra anomalies pertaining to osteoarthritis such as claw, traction and anterior osteophytes can aide in treatment plans for the patient. New size invariant features were developed for the detection of claw, traction and anterior osteophytes in cervical spine vertebrae. Using a K-means clustering and nearest centroid classification approach, the results were generated that were capable of discriminating cervical vertebrae for presence of anomalies related to osteophytes. The techniques developed can be integrated into systems based on querying spine images to be classified for such anomalies.

Computed tomography (CT) scan images of lumbar spine models are investigated and three dimensional models are generated for studying the shape and structure of the lumbar spine. Using the 3D models, techniques are developed for the detection of traction in lumbar x-ray images. Using K-means clustering and nearest centroid classification, attempts are made to classify lumbar spine images based on presence of traction.

ACKNOWLEDGEMENTS

As a prelude to this thesis I would like to express my sincere gratitude to those precious few without whose co-operation I would have been unable to engage in attempting to accomplish all of my undertaken endeavors.

At the outset I take the opportunity to thank my parents who have been a constant source of strength throughout my life. They have always allowed me to venture out all opportunities that have come across to me with all of their support. I am grateful to my brother Husain Das who has always been a source of inspiration and whose constant words of motivation have enabled me to work diligently and has taught me to face all challenges dauntlessly.

I express my sincere gratitude to Dr. Ronald Joe Stanley who has week-in and week-out spared time to guide me and help me in the comprehension, analysis and implementation of the work outlined here. Being an apprentice here, I have not only been successful in completion of the projects undertaken here but have also had the opportunity to research intensively on various aspects related medical image processing, pattern classification, software development and software integration processes. The experience and expertise that I have gained here will truly help me in all the future endeavors in my life ahead.

Special thanks go to Dr. Fikret Ercal, who being my academic advisor has guided me through out in my graduate studies and in selection of all course work. I am thankful to my thesis committee member Dr. Randy H. Moss and Dr. Bruce McMillin who have examined my research work and provided insightful suggestions.

Lastly, I am grateful to all those who have directly and indirectly helped me during the course of my studies and my research.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGEMENTS iv

TABLE OF CONTENTS v

LIST OF ILLUSTRATIONS vii

LIST OF TABLES ix

1. INTRODUCTION 1
    - 1.1. ANOMALIES PERTAINING TO THE SPINE 1
    - 1.2. THESIS OVERVIEW 6

2. SIZE-INVARIANT FEATURES FOR DISCRIMINATING CERVICAL VERTEBRAE FOR THE PRESENCE OF ANAMOLIES 7
    - 2.1. ALGORITHMS FOR CALCULATING SIZE-INVARIANT FEATURES 7
        - 2.1.1. Overview of the problem 7
        - 2.1.2. Determination of vertebral boundary 7
        - 2.1.3. Pre-processing involved towards calculation of size-invariant convex-hull based features 8
        - 2.1.4. Description of size-invariant convex-hull based features 12
        - 2.1.5. Preprocessing involved towards calculation of size-invariant features based on flipping of vertebra over centroidal axes 13
        - 2.1.6. Description of size-invariant features based on flipping of vertebra over centroidal axes 17
    - 2.2. EXPERIMENTS PERFORMED 18
        - 2.2.1. Experimental Data 18
        - 2.2.2. Generation of training and test sets 20
        - 2.2.3. Classification 20
    - 2.3. RESULTS AND CONCLUSION 23
        - 2.3.1. Experimental Results 23
        - 2.3.2. Conclusions 28

3. ORIENTATION ESTIMATION OF LUMBAR VERTEBRAE IN X-RAY IMAGES USING 3D MODELS 30

3.1. GENERATION OF 3D MODELS 30

3.1.1. Overview of the problem 30

3.1.2. Azimuth and elevation angles 32

3.1.3. Cropping the pedicle 33

3.2. CREATING AND SAVING THE PROJECTION IMAGES 38

3.2.1. Projection of 3D models 38

3.2.2. Storing and indexing of projection images 39

3.3. ALGORITHM TO COMPUTE ORIENTATION CHARACTERISTICS OF LUMBAR VERTEBRAE 41

3.3.1. Determination of vertebral boundary 41

3.3.2. A customized algorithm for resizing images 42

3.3.3. Algorithm to compute the best matching 3D projection for each lumbar vertebra 44

3.4. EXPERIMENTS PERFORMED 51

3.4.1. Experimental Data 51

3.4.2. Training data and test data 53

3.4.3. Classification 53

3.5. RESULTS AND DISCUSSION 56

3.5.1. Experimental Results 56

3.5.2. Conclusions 56

APPENDICES

A. Read-Me file for the project ‘Discrimination of Cervical Vertebrae for presence of Claw, Traction and Anterior Osteophytes’ 59

B. Read- Me file for the project ‘NewDiscSpaceNarrowing’ 76

C. Read- Me file for the project ‘NewSubluxation’ 81

BIBLIOGRAPHY 87

VITA 89

LIST OF ILLUSTRATIONS

Figure Page

1-1: Cervical spine x-ray image example from the NHANES image collection archived at National Library of Medicine (NLM). Cervical vertebrae are highlighted in the boxed region. 2

1-2: The boundary shape of cervical vertebrae extracted from x-ray images....................3

1-3: Lumbar spine x-ray image example from the NHANES image collection archived at National Library of Medicine (NLM). Lumbar vertebrae are highlighted in the boxed region. 4

1-4: The boundary shape of lumbar vertebrae extracted from x-ray images....................5

2-1: A filled cervical vertebra....................8

2-2: Filled convex hull of the cervical vertebra in Figure 2-1....................9

2-3: Exclusive-OR region between filled vertebra D and convex hull H....................10

2-4: Example of extraction of anterior, superior and inferior sides from exclusive-OR of filled vertebra and convex hull....................10

2-5: Image of cervical vertebra illustrating the posterior side bounded by dotted lines which are passing through the centroid of the vertebra....................12

2-6: Moment normalized vertebra image with the posterior side horizontal. The centroidal axis parallel to the posterior edge divides the vertebra into two halves, X_T and X_B....................15

2-7: Moment normalized vertebra image with the superior side horizontal. The centroidal axis parallel to the superior edge divides the vertebra into two halves, Y_T and $Y_{B.}$....................16

3-1: 3D model of lumbar vertebra L1 obtained by segmentation and smoothening of the layered CT scan images. 31

3-2: 3D model for lumbar vertebra L2 obtained by segmentation and smoothening of the layered CT scan images. 32

3-3: Azimuth and elevation angles [16]....................33

3-4: Top view of 3D model L1. 34

3-5: Center blob from top view of 3D model for L1. 34

3-6: 3D model L1 showing optimal pixel to cut off pedicle....................35

3-7: 3D model for lumbar vertebra L1 after cropping out the pedicle....................36

3-8: 3D model for lumbar vertebra L2 after cropping out the pedicle..............................36

3-9: 3D model for lumbar vertebra L3 after cropping out the pedicle..............................36

3-10: 3D model for lumbar vertebra L4 after cropping out the pedicle............................37

3-11: 3D model for lumbar vertebra L5 after cropping out the pedicle............................37

3-12: A projection of lumbar vertebra L1 at viewing angles (-90.0, 1.0). Lengths of posterior and superior sides are labeled. 40

3-13: A filled lumbar vertebra obtained by B-spline and image fill operation over the 36 boundary points...41

3-14: Exclusive-OR between lumbar vertebra L_R and its optimal projection. 47

LIST OF TABLES

Table Page

2-1: Distribution of cervical vertebrae dataset for detecting claw, traction and anterior osteophytes....19

2-2: K-Means classification results for cervical vertebrae using six features. 24

2-3: K-Means classification results for cervical vertebrae using seven features. 25

2-4: Number of clusters used for each classification using K-means clustering over six features (F_1-F_6). 26

2-5: Number of clusters used for each classification using K-means clustering over seven features (F_1-F_7)....27

2-6: K-Means classification results for cervical vertebrae using the five convex hull based features....28

3-1: Computation times of resizing images for the investigated resizing functions....44

3-2: Distribution of lumbar vertebrae cases based on type of lumbar vertebra. 52

3-3: Areas obtained under Receiver Operating Characteristic (ROC) curve for classifying lumbar vertebra L1-L5 for traction, respectively. 56

1. INTRODUCTION

1.1. ANOMALIES PERTAINING TO THE SPINE

Osteoarthritis is the term used to describe the deterioration of joints in the body due to age, injury or disease. Osteoarthritis affects more than 16 million people in America alone, with a higher probability of affecting people over the age of 75 years [1]. Osteoarthritis can involve the loss of the cartilage tissue between bones or joints, which can cause an increase in the friction of joints, leading to a sense of pain and over time limiting the mobility of joints. An inflammation can also occur on these joints affected by osteoarthritis which can be seen as an abnormal bone growth or bone spurs, called as osteophytes [2, 3].

The Lister Hill National Center for Biomedical Communications, an R&D division of the National Library of Medicine (NLM), National Institutes of Health (NIH), has been active in conducting research in the field of analysis of x-ray images of the spine using computer assisted techniques. It has developed a system called the Web-based Medical Information Retrieval System (WebMIRS) which provides online access to a large repository of x-ray images of spines and other associated data that were surveyed as a part of the National Health and Nutrition Examination Surveys (NHANES) [4]. Several techniques have been developed that allow researchers and other groups to retrieve such data efficiently. The conditions pertaining to the presence of osteoarthritis can be studied using digitized radiographs like x-rays and computed tomography (CT) scans obtained by Content Based Image Retrieval (CBIR) techniques. This research undertaking was devoted to the development of computer aided techniques with use of x-rays and CT scans in order to assist in the discrimination of variations of anterior osteophytes in normal cervical and lumbar spine vertebrae.

Several methods had been investigated to classify anterior osteophytes. Macnab's classification is based on radiology and pathology [5, 6], and involves a grading system defined by a medical expert to assign severity levels to the Macnab classes. Macnab's classification defines claw and traction osteophytes. A claw osteophyte extends from the vertebral rim and curves in the direction of the adjacent disc. A claw region is typically triangular in shape and is curved at the tip of the region. A traction osteophyte tends to protrude horizontally, is usually thick, does not tend to curve at the tips and does not extend across the inter-vertebral disc space. The severity grading system includes three grades for osteophytes as slight, moderate and severe. If a vertebra does not exhibit claw or traction or does not exhibit a moderate or a severe grade for anterior osteophytes, the vertebra is considered normal. These abnormalities can lead to friction between joints, deterioration of the bone tissue and the cartilage tissue around the vertebra, causing pain and also can limit the mobility of joints [2]. Hence, early detection of these anomalies can be helpful in assisting the development of patient treatment plans.

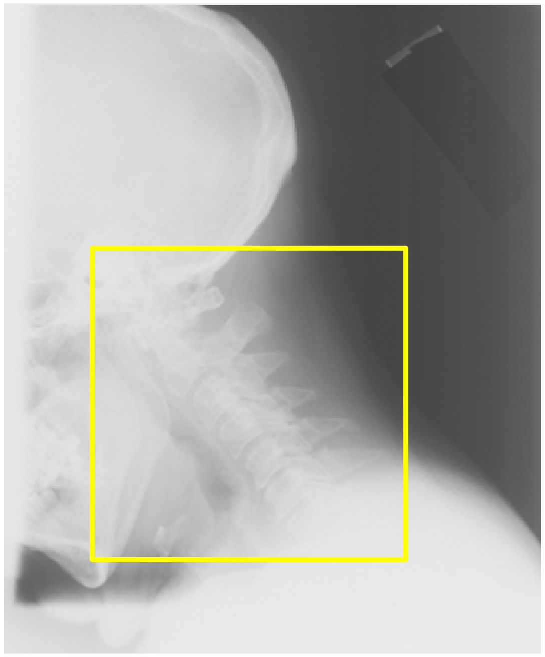

Figure 1-1: Cervical spine x-ray image example from the NHANES image collection archived at National Library of Medicine (NLM). Cervical vertebrae are highlighted in the boxed region.

Figure 1-1 presents an example of a cervical spine x-ray image. The highlighted region shows the cervical spine vertebrae. Figure 1-2 provides the borders of cervical vertebrae C3–C6, as determined by a domain expert at the National Library of Medicine (NLM). For each cervical vertebrae provided in the data set a set of 36 points on the vertebral boundary were provided which could define the shape of the vertebra. Also, for each case truth values indicative of the presence of claw, traction and anterior osteophytes were provided by a domain expert at NLM. Size-invariant descriptors based on convex hull of vertebrae had been investigated [2] to classify lumbar vertebrae for the presence of anterior osteophytes. A similar approach is adopted for classifying cervical vertebrae for the presence of claw, traction and anterior osteophytes. The shape of a normal vertebra is typically rectangular and hence similar to its convex hull, hence any deviation from its regular rectangular shape could be tagged as a presence of an anterior osteophyte. Figure 1-2 shows an example of cervical spine vertebrae where vertebra C3 shows a presence of traction and a moderate anterior osteophyte, C4 shows the presence of claw and severe anterior osteophyte, and C5 shows a presence of traction. In this research undertaking, new size-invariant descriptors are proposed for detecting claw, traction and anterior osteophytes in cervical vertebrae. The new size-invariant features proposed are based on comparing the opposite edges of the vertebra about axes passing through its centroid.

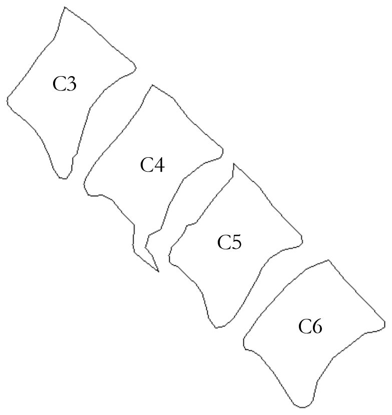

Figure 1-2: The boundary shape of cervical vertebrae extracted from x-ray images.

The use of 3D models of body joints is an active research field and can offer newer avenues to be explored by studying 3D models of vertebral joints. Three-dimensional modeling of lumbar vertebrae is discussed in this thesis which can be used in detection of various deformations relating to the vertebral spine column like traction. Traction as defined by Macnab's classification is an osteophyte that is usually thick and protrudes horizontally, which does not tend to curve at the tips and does not extend across the inter-vertebral disc space. Figure 1-3 presents an example of a lumbar spine x-ray image. The highlighted region shows the cervical spine vertebrae. Figure 1-4 provides the borders of lumbar spine vertebrae L1–L5, as determined by a domain expert at the National Library of Medicine (NLM).

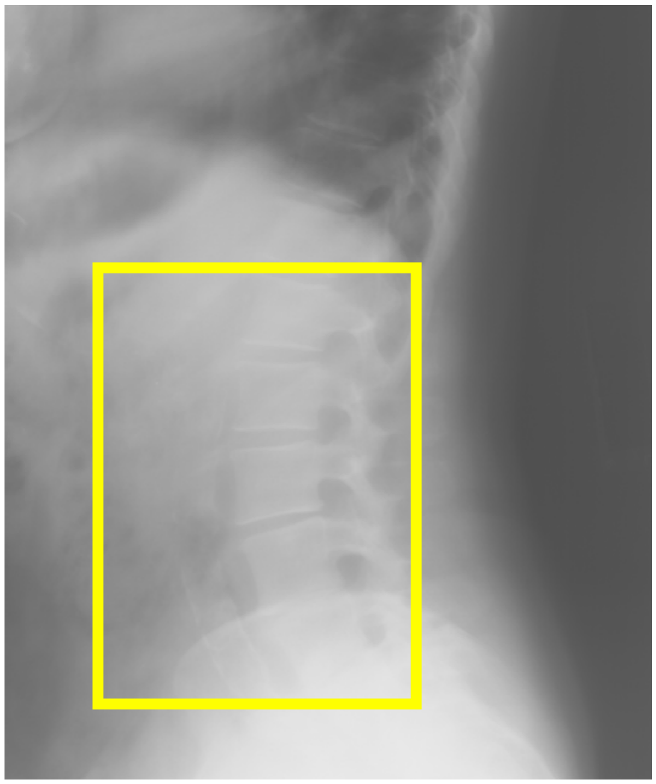

Figure 1-3: Lumbar spine x-ray image example from the NHANES image collection archived at National Library of Medicine (NLM). Lumbar vertebrae are highlighted in the boxed region.

The lumbar vertebrae are larger in size as compared to the cervical vertebrae. Hence, we investigate into generating 3-D models to assist size-invariant features in detecting anomalies in lumbar vertebrae. Three-dimensional models of lumbar vertebrae are developed for estimating two-dimensional projected model representations for lumbar vertebrae in x-ray images. The three-dimensional (3D) models of lumbar vertebrae, L1-L5, are derived from cross-sectional CT images of a normal lumbar vertebra and algorithms for combining them into 3D representations were provided by Dr. Sameer Antani and Dr. Rodney Long at NLM. A data set of lumbar vertebrae is provided with images and necessary textual data. For each lumbar vertebra provided in the data set a set of 36 points on the vertebral boundary are provided which could define the shape of the vertebra. The data set also consists of a truth table for each case indicating the presence or absence of traction as provided by a domain expert at NLM. This thesis presents the work done towards detection of traction based on K-Means clustering model development from two-dimensional projections of the 3D modes models and nearest centroid classification of lumbar spine x-ray images.

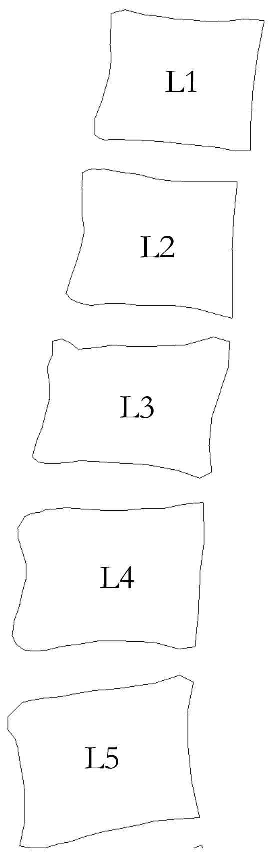

Figure 1-4: The boundary shape of lumbar vertebrae extracted from x-ray images.

1.2. THESIS OVERVIEW

This thesis introduces image analysis techniques and pattern classification methods to determine anomalies related to the vertebral spine. Previously, computer assisted techniques were studied using radius of curvature and boundary gradient features for detecting anterior osteophytes in cervical vertebrae [7] and were also used in studying herniation classification of inter vertebral discs in lumbar vertebrae [8]. In this thesis, x-ray images are used for cervical vertebrae and techniques are developed to detect anomalies and classify them accordingly. Also, the use of computed tomography scans for modeling of lumbar spine is investigated so as to allow visualizing the lumbar spine in three dimensional orientations. Attempts are made to detect the presence of traction in lumbar x-ray spine images by comparing them to projections of the 3D models.

The sections in the remainder of the thesis are explained below. Chapter 2 describes the methods used and experiments performed for discriminating cervical vertebrae for presence of claw, traction, and anterior osteophytes. Sub-sections 2.1.3 and 2.1.4 describe the procedure for calculating size-invariant features for cervical vertebrae based on the convex hull techniques and sub-sections 2.1.5 and 2.1.6 describe the procedure for calculating size-invariant features for cervical vertebrae by flipping across centroidal axes. Section 2.2 describes the experiments performed for the purpose of classifying the data set of cervical vertebrae using the size-invariant features calculated for each case. The results of the classification problem are mentioned and discussed in section 2.3. Chapter 3 investigates the use of three dimensional modeling of lumbar vertebrae in order to assist in developing methods to discriminate lumbar vertebrae based on the presence of traction. Sections 3.1 and 3.2 explain the generation of 3D models of lumbar vertebrae using computed tomography (CT) scans and the generation of the projections of 3D models at different viewing angles respectively. Section 3.3 describes an algorithm developed to compare lumbar vertebrae images with projections of 3D models at different viewing angles. The results and conclusions of the methodologies used are explained in section 3.5.

2. SIZE-INVARIANT FEATURES FOR DISCRIMINATING CERVICAL VERTEBRAE FOR THE PRESENCE OF ANAMOLIES

2.1. ALGORITHMS FOR CALCULATING SIZE-INVARIANT FEATURES

2.1.1. Overview of the problem

In this study, new size-invariant features are proposed for cervical vertebrae analysis, including anterior osteophytes discrimination and the detection of claw and traction. The proposed features extend previous research to detect anterior osteophytes [2], which utilized size-invariant-based descriptors to quantify deviations of a vertebra's shape from its typical convex shape. This study proposes new size-invariant descriptors beyond the analysis of convex hulls.

2.1.2. Determination of vertebral boundary

For each cervical vertebra in the data set we are provided with a text file which consists of (x, y) pair of coordinates of 36 points along the boundary of the vertebra. These 36 points are marked along the vertebra boundary by experienced radiologists and domain experts. For the purpose of calculating the size-invariant features, we need to compute the shape of the vertebra. A second order B-spline [2] algorithm was applied to the set of 36 coordinates that computes a set of connected points which make up the complete vertebra boundary. An image fill operation was performed upon the set of connected boundary pixels to get the completely filled vertebra. If $D = D(x, y)$ denotes the filled vertebra, then D was defined by equation 2.1. Figure 2-1 shows a filled cervical vertebra to illustrate the procedure explained above.

$$D = \begin{cases} 1, & \textit{if } (x, y) \textit{ lies inside or on the vertebra boundary} \\ 0, & \textit{elsewhere.} \end{cases} \tag{2.1}$$

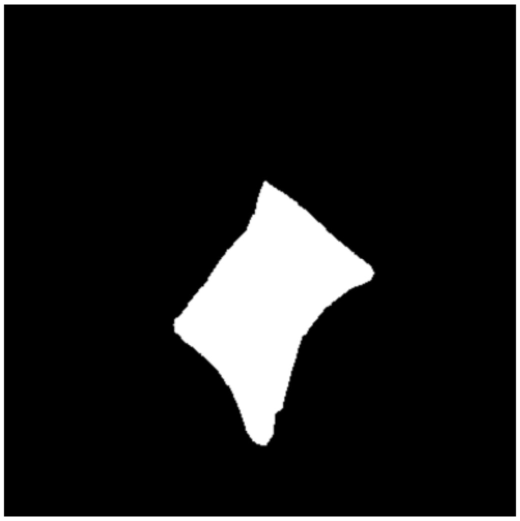

Figure 2-1: A filled cervical vertebra.

2.1.3. Pre-processing involved towards calculation of size-invariant convex-hull based features

The first five features developed for discrimination of claw, traction and anterior osteophytes were based on comparisons between vertebra image and the convex hull of the vertebra [3]. The convex hull of a set of points Q is defined as the smallest convex simple polygon enclosing all the points of Q [9]. In order to compute the convex hull of the vertebra image, we use the implementation of the Quickhull algorithm provided by Barber et al. [10]. The convex hull of the cervical vertebra in Figure 2-1 is shown in Figure 2-2. The convex hull image $H = H(x, y)$ is defined as,

$$H = \begin{cases} 1, & \text{if } (x, y) \text{ lies inside the convex hull of } D \\ 0, & \text{elsewhere.} \end{cases} \tag{2.2}$$

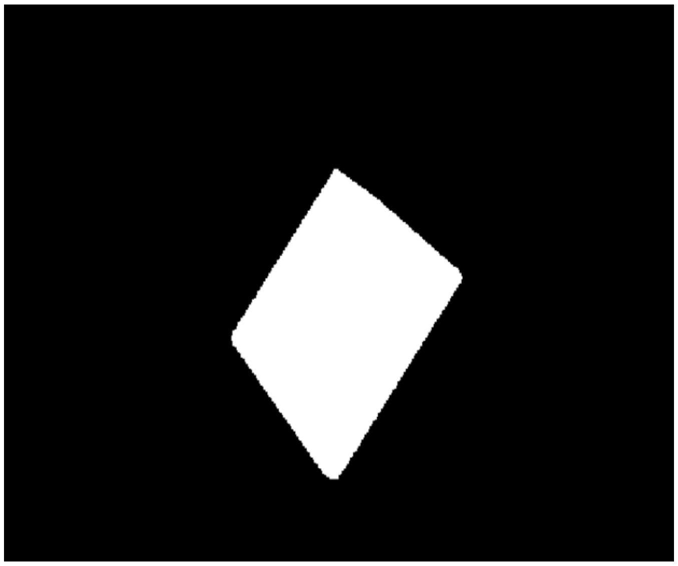
Figure 2-2: Filled convex hull of the cervical vertebra in Figure 2-1.

Let X be the exclusive-OR of the vertebra image D with the filled convex hull image H. Figure 2-3 shows the exclusive-OR of the cervical vertebra shown in Figure 2-1 with its convex hull. Figure 2-4 describes the areas obtained in X pertaining to the different edges of the vertebra. The areas pertaining to the superior side, inferior side and the anterior side are labeled.

$$X(x, y) = D(x, y) \oplus H(x, y) \tag{2.3}$$

$$X(x, y) = \begin{cases} 0, & \text{if } D(x, y) = H(x, y) = 1 \vee D(x, y) = H(x, y) = 0 \\ 1, & \text{otherwise} \end{cases} \tag{2.4}$$

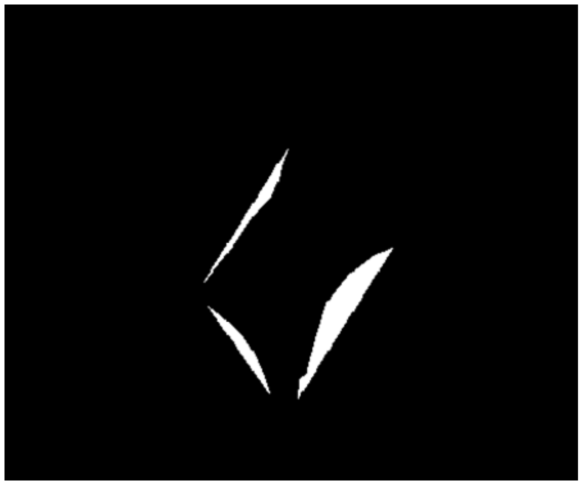

Figure 2-3: Exclusive-OR region between filled vertebra D and convex hull H.

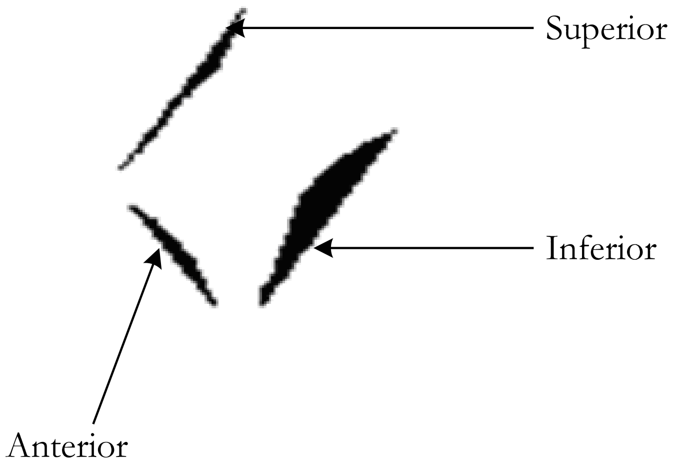

Figure 2-4: Example of extraction of anterior, superior and inferior sides from exclusive-OR of filled vertebra and convex hull.

Depending upon the vertebra under consider, X may include one or more connected regions for each vertebral side showing a concave edge for the corresponding vertebra. Let A_D, A_H *and* A_X be the areas of vertebra image D, convex hull region H, and the exclusive-OR between D and H. Let k denote the number of distinct connected regions in X considering 8-connectivity. If X_i, for $i = 1, 2, 3, ..., k$, denotes each of the distinct connected regions, then the set X can be represented as $X = \bigcup_{i=1}^{k} X_i$. The centroid $(\bar{x}_D, \bar{y}_D)$ of the filled vertebra D was calculated and the centroids $(\bar{x}_i, \bar{y}_i)$, $i = 1, 2, 3, ..., k$, for each of the k connected components within X having areas $A_1, A_2, A_3, ..., A_k$ were calculated. Figure 2-5 shows the position of the centroid of the cervical vertebra in Figure 2-1. Since, this research undertaking was aimed at calculating abnormalities like claw, traction and osteophytes pertaining to the anterior side of vertebrae, it can be concluded that the information corresponding to the posterior region does not contribute in discriminating such anterior side abnormalities and hence can be considered irrelevant. Consider $X_{\bar{P}}$ to be the subset of set X, $X_{\bar{P}} \subseteq X$; where $X_{\bar{P}}$ was computed as $X_{\bar{P}} = \bigcup_{i=1}^{k} X_m$ such that X_m does not belong to the posterior side of the vertebra. In order to identify if X_i belongs to the posterior side or not, we consider the positioning of the centroid $(\bar{x}_i, \bar{y}_i)$ corresponding to region X_i. The region X_i was said to belong to the posterior side if $\bar{x}_i > \bar{x}_D$ and $\bar{y}_i < \bar{y}_D$. Such X_i are not included in the set $X_{\bar{P}}$. The area of the exclusive-OR region not including the posterior side was given by, $A_{\bar{P}} = \sum_{i=1}^{k} A_i$, for all i, such that, $1 \le i \le k$ and $X_i \subseteq X_{\bar{P}}$. In order to compute the area of pertaining only to the inferior side, we consider the subset X_I of X, $X_I \subseteq X$, where $X_I = \bigcup_{i=1}^{k} X_m$, such that X_m belongs to the inferior side of the vertebra. Analogous to computing $X_{\bar{P}}$, any region X_i was considered in computing X_I, if and only if, for the centroid $(\bar{x}_i, \bar{y}_i)$ corresponding to region X_i, the condition $\bar{x}_i > \bar{x}_D$ and $\bar{y}_i > \bar{y}_D$ was true. The area of the exclusive-OR region pertaining only to the inferior side of the vertebra was given as, $A_I = \sum_{i=1}^{k} A_i$, such that, $1 \le i \le k$ and $X_i \subseteq X_I$. Similarly, the areas corresponding to the superior side of the vertebra, A_S and the anterior side of the vertebra, A_T were calculated. The calculated values for areas were used in computing the convex-hull based features for discrimination of claw, traction and anterior osteophytes in cervical spine.

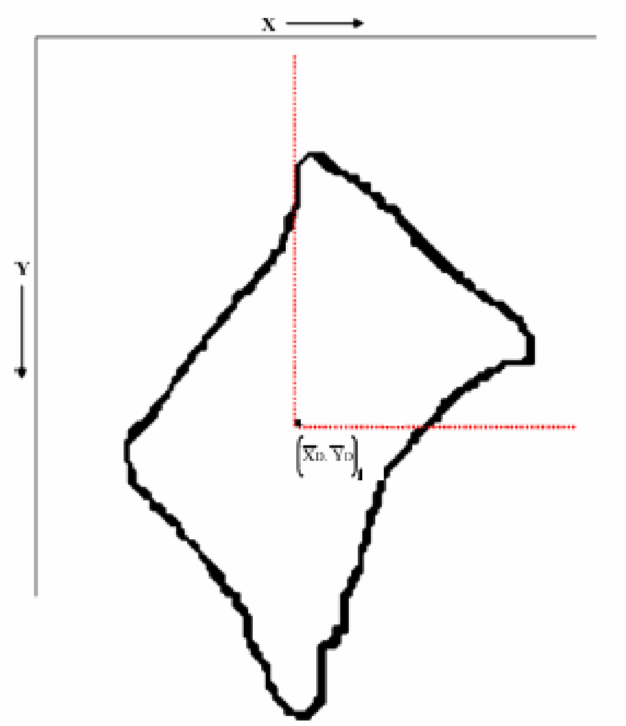

Figure 2-5: Image of cervical vertebra illustrating the posterior side bounded by dotted lines which are passing through the centroid of the vertebra.

2.1.4. Description of size-invariant convex-hull based features

For a given vertebra, the following features were calculated based on the computations described in sub-section 2.1.3:

1) The ratio between the area of the filled vertebra and the area of the filled convex hull of the vertebra,

$$F_1 = \frac{A_D}{A_H} \tag{2.5}$$

2) The ratio between the area of the exclusive-OR region without the posterior side regions and the area of the filled convex hull of the vertebra,

$$F_2 = \frac{A_{X_{\bar{P}}}}{A_H} \tag{2.6}$$

3) The ratio between the area of the exclusive-OR regions pertaining to the inferior side of the vertebra and the area of the vertebra,

$$F_3 = \frac{A_I}{A_D} \tag{2.7}$$

4) The ratio between the area of the exclusive-OR regions pertaining to the superior side of the vertebra and the area of the vertebra,

$$F_4 = \frac{A_S}{A_D} \tag{2.8}$$

5) The ratio between the area of the exclusive-OR regions pertaining to the anterior side of the vertebra and the area of the vertebra,

$$F_5 = \frac{A_T}{A_D} \tag{2.9}$$

2.1.5. Preprocessing involved towards calculation of size-invariant features based on flipping of vertebra over centroidal axes

The next features that were computed were based on flipping the vertebra about its centroidal axes. For this, first we consider the set of points D corresponding to the completely filled vertebra as described in sub-section 2.1.1. The orientation of the vertebra was estimated by computing the corner points of vertebra denoted by D. The corner points can be computed from the topmost, leftmost, rightmost and bottommost points of the vertebra.

In order to make a comparison of the shapes of the posterior and anterior sides, we first calculate the orientation of the vertebra along the posterior side by calculating the slope of the line joining the end points of the posterior side. The corners points or the end points of the posterior side correspond to the topmost and rightmost points of the vertebra as can be seen in Figure 2-1. Now, the vertebra was rotated by the angle calculated from the slope of the posterior side, such that the slope of the posterior side of the rotated vertebra becomes zero, giving the vertebra a horizontal orientation along its posterior side.

Moment normalization [11] was applied to the rotated vertebra so as to eliminate any skeweness from its alignment as shown in Figure 2-6. Let X_{MN} denote the set of points contained in the moment normalized vertebra with A_{MN} corresponding to the area described by the points in X_{MN}. Next, the centroid $(\bar{x}_{MN}, \bar{y}_{MN})$ for the moment normalized vertebra X_{MN} was calculated. Using the centroid $(\bar{x}_{MN}, \bar{y}_{MN})$, the set X_{MN} was divided into two disjoint sets X_T and X_B corresponding to top and bottom halves of X_{MN}, such that, for any point $P(x, y)$ in the set X_{MN}, $P(x, y)$ belongs to set X_T if $y \le \bar{y}_{MN}$ and $P(x, y)$ belongs to the set X_B if $y > \bar{y}_{MN}$. It can be easily seen that $X_T \cup X_B = X_{MN}$. Figure 2-6 clearly shows the sets X_{MN}, X_T and X_B. As seen in Figure 2-6, X_T denotes the posterior half of the vertebra and X_B denotes the anterior half of the moment normalized vertebra in X_{MN}.

$$X_T = \{\, P(x,y) \mid P \in X_{MN} \wedge y \le \bar{y}_{MN} \,\} \tag{2.10}$$

$$X_B = \{\, P(x,y) \mid P \in X_{MN} \wedge y > \bar{y}_{MN} \,\} \tag{2.11}$$

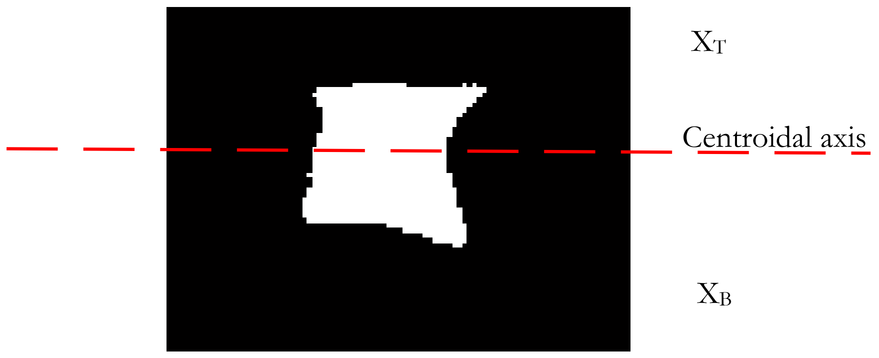

Figure 2-6: Moment normalized vertebra image with the posterior side horizontal. The centroidal axis parallel to the posterior edge divides the vertebra into two halves, X_T and X_B.

In an attempt to make a comparison between the shapes of the posterior and anterior edges, we first flip the anterior half of the vertebra in X_B along the centroidal axis of the moment normalized vertebra passing through the centroid $(\bar{x}_{MN}, \bar{y}_{MN})$ and parallel to line joining the end points of the posterior edge. Let the flipped anterior half be denoted by $X_{B\_flipped}$. Finally, we compute the set X_R as the exclusive-OR between X_T, the posterior half and $X_{B\_flipped}$, the flipped anterior half of the vertebra.

$$X_R = X_T \oplus X_{B\_flipped} \tag{2.12}$$

Let A_{R_X} be the area of the exclusive-OR set X_R.

Similarly, in order to make comparisons between the edges of the vertebra pertaining to the superior and inferior sides, we compute the orientation of the vertebra along the superior side by calculating the slope of the line joining the end points of the superior side of the vertebra. Using, the angle calculated from this slope, the vertebra was rotated such the superior side of the vertebra has a horizontal alignment. Moment normalization was performed so as to obtain the moment normalized vertebra Y_{MN}. The set Y_{MN} and the set X_{MN} correspond to the same vertebra, but are different as the orientation of the vertebra in each case differs, although they have approximately the same area A_{MN}. Next, the points in set Y_{MN} were divided into two distinct sets Y_T and Y_B, such points lying above the centroid of Y_{MN} belong to set Y_T, or set Y_B otherwise. This is done by computing a line passing through the centroid $(\bar{x}_{MN}, \bar{y}_{MN})$ of the vertebra and is parallel to the line joining the end points of the superior side. The set Y_T corresponds to the vertebra half containing the superior side and the set Y_B corresponds to the vertebra half containing the inferior side.

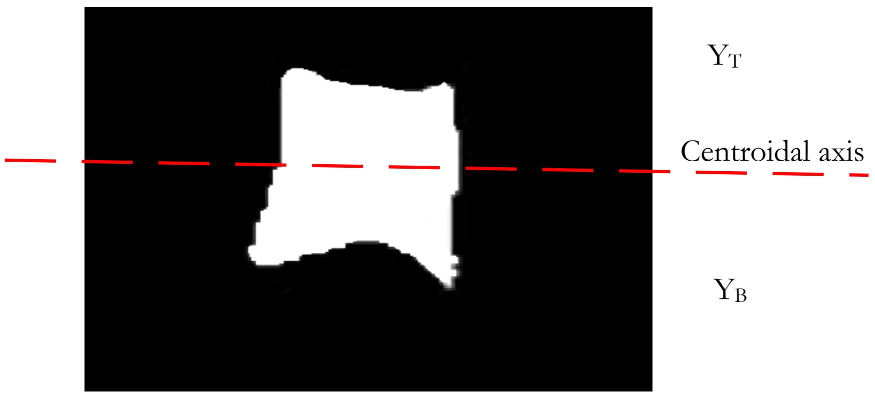

Figure 2-7: Moment normalized vertebra image with the superior side horizontal. The centroidal axis parallel to the superior edge divides the vertebra into two halves, Y_T and Y_B.

In order to compare the shapes of the superior and posterior sides, we first flip the vertebra half Y_B containing inferior side about the centroidal axis parallel to the superior side as shown in Figure 2-7. Let the flipped inferior side vertebra half be denoted by $Y_{B\_flipped}$. Finally, we compute the set Y_R as the exclusive-OR between Y_T, the vertebra half containing the superior side and $Y_{B\_flipped}$, the vertebra half containing the flipped inferior side.

$$Y_R = Y_T \oplus Y_{B\_flipped} \tag{2.13}$$

Let A_{RY} be area of the exclusive-OR set Y_R.

The calculated values for areas were used in computing novel features for discrimination of claw, traction and anterior osteophytes in cervical spine.

2.1.6. Description of size-invariant features based on flipping of vertebra over centroidal axes

For the given vertebra, the following features were calculated based on the computations described in sub-section 2.1.5:

1) The ratio of the area obtained by exclusive-OR operation between the anterior and the posterior sides of the moment normalized vertebra and the area of the moment normalized vertebra,

$$F_6 = \frac{A_{R_X}}{A_{MN}} \tag{2.14}$$

2) The ratio of the area obtained by an exclusive-OR operation between the superior and the inferior sides of the moment normalized vertebra and the area of the moment normalized vertebra,

$$F_7 = \frac{A_{R_Y}}{A_{MN}} \tag{2.15}$$

2.2. EXPERIMENTS PERFORMED

2.2.1. Experimental Data

The experimental data was provided by the National Library of Medicine (NLM), which contained the following:

1) A data sheet consisting of a table where each row was a tuple τ, $\tau = (name, c_S, c_I, t_S, t_I, o_S, o_I)$. Here, the attribute $name$ contained a string for the vertebra name. The attributes c_I and c_S have values $true/false$ indicating the presence of claw on the superior and inferior sides of the vertebra respectively. The attributes t_S and t_I have values $true/false$ indicating the presence of traction on the superior and inferior sides of the vertebra respectively. Lastly, the attributes o_S and o_I have enumerated labels $\{slight, moderate, severe\}$ indicating a grade for the presence of anterior osteophytes on the superior and inferior sides of the vertebra.
2) For each vertebra in the data sheet, a text file was provided which contained values representing (x, y) coordinates of 36 points along the vertebral boundary for the corresponding vertebra.

The data set provided consisted of a total of 390 cervical vertebrae for which the proposed features were calculated in order to facilitate in determining the presence of claw, traction and anterior osteophytes. The 36 points along the vertebral boundary for each vertebra were provided to NLM by experienced radiologists and domain experts. For the entire dataset, three new classes of attributes (c, t, o) were introduced, of which c and t had values labeled $true/false$, where c was indicative of the presence of claw and t was indicative of the presence of traction. The attribute class, c, indicating the presence of claw was assigned a value $true$, if either c_s, the attribute class for presence of claw at superior side or c_I, the attribute class for presence of claw at the inferior side had a value $true$; otherwise it was assigned the value $false$. Similarly, the attribute class

t for presence of traction was assigned values based on the values t_S and t_I corresponding to the superior and inferior sides. The attribute o had an enumerated label $\{slight, moderate, severe\}$ indicating the presence of anterior osteophytes. This attribute was assigned a value labeled $severe$ if either, o_S, the superior side or o_I, the inferior side label had a value $severe$; else it was assigned a value labeled $moderate$ if either o_S or o_I had a value $moderate$; otherwise it was assigned a value labeled $slight$. For this research undertaking, the discrimination of anterior osteophytes was done for a normal/abnormal classification. Hence, for all the vertebra cases provided in the data set, the vertebra labels for anterior osteophytes bearing a label $slight$ were considered to be normal and vertebra labels having the label $moderate$ or $severe$ were considered to be abnormal.

The data set was stratified by the type of cervical vertebrae, which are C3 – C7. It was observed that the data set of 390 cervical vertebra consisted of 97 C3s, 99 C4s, 96 C5s, 76 C6s and 22 C7s. The 390 entries in the data set when grouped by target variables c, t and o showed the following distribution.

Table 2-1: Distribution of cervical vertebrae dataset for detecting claw, traction and anterior osteophytes.

	Number of cervical vertebrae
Claw/No claw	242/148
Traction/No traction	212/178
Anterior Osteophytes (abnormal/normal)	258/132 (82 $severe$, 176 $moderate$)

The features $F_1 - F_7$ explained in section 2.1 are calculated for each vertebra provided in the data set. These features were developed keeping in mind the aim of this research undertaking which was investigation of cervical vertebra for the presence claw, traction and anterior osteophytes that have any characteristic deviations in their shape from a normal rectangular shape. The features $F_1 - F_5$ which are based on comparisons with the convex hull of the vertebra and the features $F_6 - F_7$ which characterize the difference in curvature of the sides of the vertebra can be considered as the basis for detection of claw, traction and anterior osteophytes.

2.2.2. Generation of training and test sets

In order to generate the training and test sets, first we integrated the features calculated for each vertebra and the data provided. The integrated data set hence consisted of tuples of the form, $\tau_n = (name, F_1, F_2, F_3, F_4, F_5, F_6, F_7, c, t, o)$; where $name$, c, t and o are as explained in sub-section 2.2.1 and $F_1 - F_7$ are the features calculated for the vertebra corresponding to $name$.

For classification of cervical vertebra for the presence of claw as claw/no claw, for the presence of traction as traction/no traction and the presence of anterior osteophytes as abnormal/normal, twenty randomly generated training sets and test sets were generated for each of the three classification problems. The data set was divided into training and the test sets. Ninety percent of the normal and abnormal feature vectors were used in the training set and the remaining ten percent for the test set.

2.2.3. Classification

The three classification problems involved generating a model that could classify a given input vector into classes claw/no claw, traction/no traction, and abnormal/normal osteophytes, respectively. These classifications were performed on the data set of cervical vertebrae with features $F_1 - F_6$ and with features $F_1 - F_7$ separately. The following procedure was applied for each of the three classification problems (claw/no claw, traction/no traction and abnormal/normal osteophytes). For each of the 20 randomly generated training and test sets, first, the mean and standard deviation values, μ and σ were calculated for all features F in the feature set $F_1 - F_7$ of the training set.

Second, the feature vectors are normalized by subtracting each feature by its mean and dividing by its standard deviation. For each feature F in the feature set, we calculate the normalized feature F_{norm} as, $F_{norm} = \{f_{norm} \mid \forall f \in F, \exists f_{norm}\}$, where f_{norm} was calculated as, $f_{norm} = \frac{f - \mu}{\sigma}$, for $f_{norm} \in F_{norm}$ and $f \in F$. Third, the number of clusters for each class (claw, no claw) was estimated by using subtractive clustering [12, 13]. Fourth, using the normalized featured vectors for the training data and the number of cluster estimated for each class, K-means clustering [14, 15] was performed to determine the cluster centers for each class. Fifth, we normalize the set of test vectors using the mean and standard deviation values obtained for the training set. Sixth, for each of the feature vectors in the test set, nearest centroid classification was performed. For each normalized feature vector in the test set taken, the Euclidean distance to the cluster centers of each class were computed. The minimum of the Euclidean distance was calculated and depending upon the class (claw, no claw) of the cluster center for which the Euclidean distance was of minimum value, a similar label was assigned to the test feature vector. Seventh, the true negative and true positive classification rates are computed for the test data. True positive refers to the percentage of test case vertebrae with claw being classified correctly and true negative refers to the percentage of test case vertebrae with no claw being classified correctly. Eighth, the process was repeated for all the 20 randomly generated training and test sets. The entire procedure of classification was performed over the set of features $F_1 - F_6$ and the set of features $F_1 - F_7$ and corresponding results were generated.

The procedure for classification of traction and anterior osteophytes was analogous to the procedure for claw. The cluster centers for the classes of traction and no traction are computed in the process of classifying traction. The procedure for classifying anterior osteophytes was slightly modified. For anterior osteophytes, cluster centers were calculated for each of the three classes slight, moderate and severe. For the test vectors, Euclidean distances were computed to each of the cluster centers. If the minimum Euclidean distance corresponded to a cluster center for class slight, then the test vector was assigned the label normal. If the minimum Euclidean distance corresponded to a cluster center for either the class moderate or the class severe, the test vector was assigned a label abnormal.

2.3. RESULTS AND CONCLUSION

2.3.1. Experimental Results

The results obtained from the experiments performed to classify cervical vertebrae as claw/no claw, traction/no traction, abnormal/normal for anterior osteophytes computed based on the six features and computed based on seven features are discussed below. The classification was done over the provided data set of 390 cervical vertebrae. Table 2-2 below contains the results of the experiments performed using the six features $F_1 - F_6$ for classification of claw/no claw, traction/no traction and abnormal/normal for anterior osteophytes for cervical vertebrae. Table 2-3 below contains the results of the experiments performed using the seven features $F_1 - F_7$ for classification of claw/no claw, traction/no traction and abnormal/normal for anterior osteophytes for cervical vertebrae as done for the six features $F_1 - F_6$. Table 2-2 and Table 2-3 show the results of classifying the 20 randomly generated test sets using clustering techniques over the classification models obtained for the training sets using six and seven features respectively as discussed in sub-section 2.2.3.

In Table 2-2 and Table 2-3, the column 1 gives the iteration of the training and test sets generated. Columns 2 and 3 provide the results obtained for claw/no claw classification, Columns 4 and 5 contain the results obtained for traction/no traction classification and Columns 6 and 7 give the results for abnormal/normal classification for detection of anterior osteophytes. All the vertebrae bearing grades moderate or severe were considered abnormal and all the vertebrae bearing grades slight were considered normal. Also note that 20 different training and test sets were generated for each classification problem. The mean and standard deviation values for each classification result were found are shown at end of Table 2-2 and Table 2-3.

Table 2-2: K-Means classification results for cervical vertebrae using six features.

Iter.	% Correct Claw	% Correct No Claw	% Correct Traction	% Correct No Traction	% Correct Abnormal	% Correct Normal
1	84.00	71.43	91.67	80.00	85.19	83.33
2	88.00	78.57	79.17	100.00	92.59	66.67
3	88.00	64.29	83.33	86.67	96.30	66.67
4	80.00	78.58	87.50	93.33	77.78	83.33
5	84.00	71.43	83.33	93.33	81.48	83.33
6	80.00	78.58	100.00	73.33	92.59	66.67
7	84.00	92.86	79.17	100.00	85.19	83.33
8	96.00	57.14	79.17	100.00	81.48	66.67
9	88.00	71.43	83.33	86.67	85.19	66.67
10	88.00	78.58	87.50	80.00	85.19	66.67
11	92.00	64.29	91.67	73.33	81.48	83.33
12	80.00	78.58	83.33	86.67	92.59	83.33
13	84.00	71.43	83.33	86.67	77.78	75.00
14	80.00	78.57	83.33	86.67	85.19	83.33
15	84.00	85.72	83.33	86.67	77.78	83.33
16	96.00	57.14	83.33	86.67	85.19	83.33
17	76.00	85.71	91.67	66.67	81.48	75.00
18	92.00	71.43	87.50	73.33	74.07	91.67
19	84.00	85.72	83.33	80.00	81.48	83.33
20	88.00	64.29	95.83	60.00	85.19	66.67
Mean	85.80	74.29	86.04	84.00	84.44	77.08
Std.Dev.	5.43	9.67	5.62	10.90	5.84	8.50

Table 2-3: K-Means classification results for cervical vertebrae using seven features.

Iter.	% Correct Claw	% Correct No Claw	% Correct Traction	% Correct No Traction	% Correct Abnormal	% Correct Normal
1	84.00	71.43	87.50	80.00	76.92	83.33
2	92.00	78.57	83.33	93.33	73.07	66.67
3	80.00	64.29	79.17	93.33	69.23	100.00
4	72.00	78.58	79.17	93.33	53.84	76.92
5	80.00	85.72	83.33	86.67	76.92	84.62
6	88.00	71.43	83.33	86.67	72.00	72.73
7	88.00	85.72	70.83	93.33	69.23	81.82
8	96.00	50.00	87.50	66.67	80.77	70.00
9	84.00	71.43	79.17	86.67	76.92	66.67
10	92.00	78.58	87.50	80.00	76.92	90.91
11	80.00	64.29	79.17	80.00	73.08	69.23
12	72.00	78.58	87.50	93.33	84.62	100.00
13	80.00	85.72	87.50	80.00	84.62	69.23
14	88.00	71.43	95.83	73.33	69.23	91.67
15	88.00	85.72	83.33	80.00	69.23	80.00
16	96.00	50.00	75.00	80.00	65.38	76.92
17	84.00	92.86	75.00	80.00	76.00	54.55
18	88.00	71.43	79.17	86.67	80.77	83.33
19	88.00	85.72	70.83	80.00	84.62	81.82
20	88.00	57.14	91.67	100.00	73.08	63.64
Mean	85.40	73.93	82.29	84.67	74.32	78.20
Std.Dev.	6.41	11.75	6.36	7.86	7.22	11.65

Table 2-4 and Table 2-5 provide the number of clusters determined by the subtractive clustering used over the training data set of six features and seven features respectively. The number of clusters were determined for training data sets for each classification task of claw/no claw, traction/no traction and abnormal/normal for anterior osteophytes separately as the training data sets in each task differed. This process was done for all iterations of K-means clustering and nearest centroid classification of the 20 randomly generated training sets.

Table 2-4: Number of clusters used for each classification using K-means clustering over six features (F_1-F_6).

Iter.	Number of Clusters						
	Claw	No Claw	Traction	No Traction	Severe osteophytes	Moderate osteophytes	Slight osteophytes
1	4	5	3	6	8	4	4
2	4	8	3	5	8	3	3
3	4	6	3	6	6	4	3
4	4	5	3	5	5	4	3
5	4	7	3	6	8	4	5
6	4	6	3	8	7	4	3
7	5	6	3	5	8	5	4
8	4	7	3	5	6	3	4
9	4	5	4	5	9	3	4
10	4	8	3	8	6	3	5
11	4	6	3	5	8	3	4
12	4	5	3	5	6	3	4
13	4	7	4	6	6	6	4
14	4	6	3	5	6	4	4
15	5	6	3	5	7	3	4
16	4	7	3	5	7	4	3
17	5	7	2	5	6	3	3
18	5	8	2	5	8	4	4
19	4	8	3	7	7	4	3
20	4	6	3	8	7	3	3

Table 2-5: Number of clusters used for each classification using K-means clustering over seven features (F_1-F_7).

Iter.	Number of Clusters						
	Claw	No Claw	Traction	No Traction	Severe osteophytes	Moderate osteophytes	Slight osteophytes
1	4	4	4	11	10	7	5
2	4	5	4	11	10	7	4
3	3	5	5	11	11	6	4
4	4	4	5	10	9	5	4
5	4	6	3	12	12	6	5
6	4	4	4	10	13	6	4
7	4	6	4	10	10	6	4
8	3	4	5	10	11	4	5
9	4	4	5	9	10	6	6
10	4	5	4	9	10	5	5
11	3	5	3	9	11	4	4
12	4	4	4	10	9	5	4
13	4	6	4	11	11	8	4
14	4	4	3	12	10	7	4
15	4	6	4	9	12	7	4
16	3	4	4	10	11	5	6
17	4	5	4	11	10	8	4
18	4	6	4	11	10	7	5
19	4	5	5	12	10	5	7
20	4	5	4	10	20	6	5

In order to discuss the experimental results obtained using the six features and the seven features, the results for the five convex hull features are generated so as to provide a basis to discuss the contribution of features F_6 and F_7 in the classification process. A summary of the results obtained for classification of claw, traction and anterior osteophytes using the five convex hull based features are given in Table 2-6. Columns 2 and 3 represent the results obtained for claw/no claw classification, Columns 4 and 5 represent the results obtained for traction/no traction classification and Columns 6 and 7 represent the results for abnormal/normal classification for detection of anterior osteophytes. All the vertebrae bearing grades moderate or severe were considered abnormal and all the vertebrae bearing grades slight were considered normal. Columns 2-7 provide the average results of percentage of cervical vertebrae in the data set that were classified correctly for each class.

Table 2-6: K-Means classification results for cervical vertebrae using the five convex hull based features.

	% Correct Claw	% Correct No Claw	% Correct Traction	% Correct No Traction	% Correct Abnormal	% Correct Normal
Mean	85.20	70.70	81.30	78.00	86.30	65.80
Std.Dev.	7.90	13.31	7.09	10.84	6.37	10.44

2.3.2. Conclusions

New size invariant features were investigated and developed in order to improve the results of classification for claw, traction and anterior osteophytes. It can be concluded that the proposed size-invariant features show that they are capable of discriminating cervical vertebrae for the presence of claw, traction and osteophytes as seen by the results obtained in Table 2-2 and Table 2-3. The use of features $F_1 - F_6$ provided average discrimination rates of 85.80% for claw, 86.04% for traction and 84.44% for detecting anterior osteophytes and 74.29% for no claw, 84.00% for no traction and 77.08% for normal vertebra with slight osteophyte. The use of features $F_1 - F_7$ provided average discrimination rates of 85.40% for claw, 82.29% for traction and 74.32% for detecting anterior osteophytes and 73.93% for no claw, 84.67% for no traction and 78.20% for normal vertebra with slight osteophyte. Overall, the performance compared to the results seen earlier for the five convex hull based features have been improved. This leads to the fact that the features F_6 and F_7 provide novel information in classification of cervical vertebrae for anomalies like claw, traction and anterior osteophytes.

It can be seen that the six features and the seven features provided better results for traction as compared to claw and anterior osteophytes. The results obtained by using the features F_6 and F_7 are very identical in discrimination of cervical vertebrae for presence of claw. For the case of discrimination of traction, the six features provided better results for the case where traction is correctly detected, that is the true positive cases, while the seven features provided slightly better results to detect the absence of traction correctly, that is the true negative cases. The seven features also provided better standard deviation values in detecting absence of traction. In the case of discriminating cervical vertebrae for anterior osteophytes, the six features provided far better results than the seven features. Another important observation to be made from Table 2-4 and Table 2-5 is that the number of clusters required in the process of classification of anterior osteophytes using the seven features $F_1 - F_7$ is far greater than required for the six features $F_1 - F_6$. This can be one of the reasons that better results were obtained with six features for discriminating anterior osteophytes.

The overall goal of the research undertaking was to investigate and develop features characteristic to anomalies relating to osteoarthritis such as claw, traction and anterior osteophytes in cervical vertebrae and to develop techniques to classify them accordingly. It can be concluded that the proposed features can be incorporated into a content based image retrieval (CBIR) system to allow querying of images with conditions specific to anomalies like claw, traction and anterior osteophytes.

3. ORIENTATION ESTIMATION OF LUMBAR VERTEBRAE IN X-RAY IMAGES USING 3D MODELS

3.1. GENERATION OF 3D MODELS

3.1.1. Overview of the problem

This research proposes the use of 3D models to study the shape of lumbar spine vertebrae in order to assist in detection of anomalies like traction. This involves generating methods to create 3D models that can be studied and to develop techniques using size-invariant features for classification of lumbar spine vertebra images based on presence of traction.

The initial data that was provided by NLM consisted of a series of images for each lumbar vertebra L1−L5 developed using computed tomography (CT) scans and a set of algorithms implemented in Matlab®. These images and the initial algorithms for model generation were provided by the National Library of Medicine (NLM). The implementation of the provided algorithms performed as follows. First, for each lumbar vertebra L1−L5, the corresponding CT scan images were processed using segmentation tools and a set of binary images B was generated for each L1−L5. Each of these two-dimensional binary images b, $b \in B$, corresponded to the image of the vertebra when sliced. Since, the binary images b represent the images of a sliced vertebra, hence, when these two-dimensional binary images b, $b \in B$, obtained from the layered CT scans are stacked one over the other, we get a three-dimensional structure J that describes the shape of the lumbar vertebra. Third, a three dimensional smoothing operation was performed over J to produce a smoothened shape of the vertebra, J_s. Last, the set of points in J_s are applied a *patch* routine available in Matlab®. The *patch* routine displays the points J_s to give a 3D structure which can be viewed as a solid object. The output of the *patch* routine was the required 3D model D_i as defined in equation 3.1.

$$D_i(x, y, z) = \begin{cases} 1, & \text{if } (x, y, z) \text{ is a point inside or on the structure} \\ & \text{of the lumbar vertebra.} \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

The above procedure was repeated for all five sets of binary images, B, generated for the five lumbar vertebrae L1−L5. Hence, we get a set of 3D models $D = \{D_i \mid i = 1, 2, 3, 4, 5\}$ corresponding to each of the five lumbar vertebrae. The 3D models for lumbar vertebrae L1 and L2 are shown in Figure 3-1 and Figure 3-2, respectively.

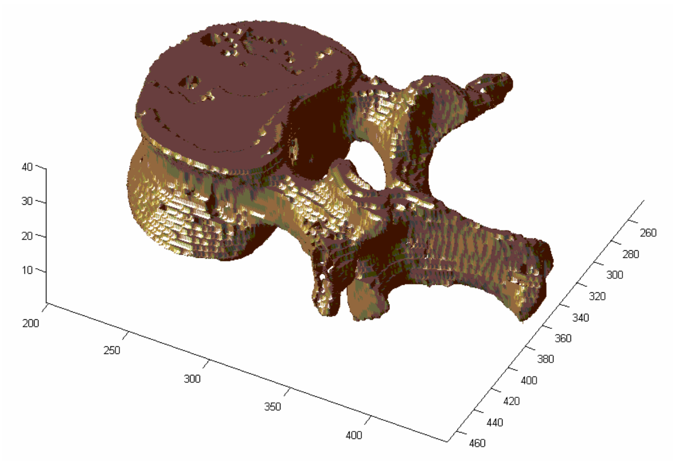

Figure 3-1: 3D model of lumbar vertebra L1 obtained by segmentation and smoothening of the layered CT scan images.

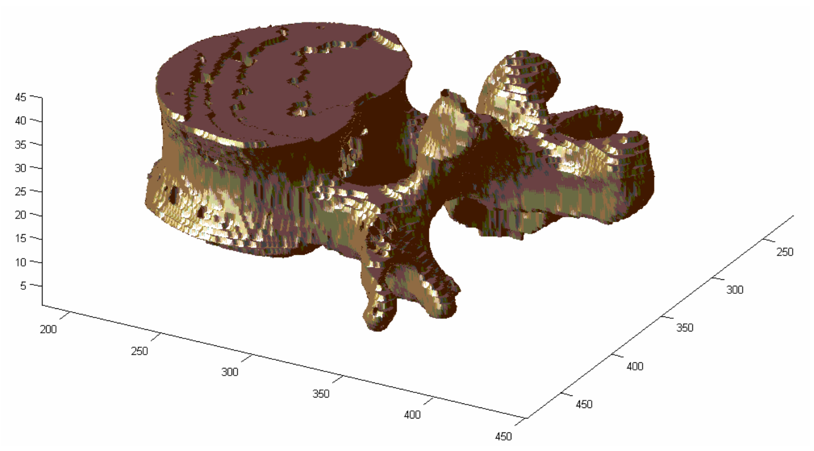

Figure 3-2: 3D model for lumbar vertebra L2 obtained by segmentation and smoothening of the layered CT scan images.

3.1.2. Azimuth and elevation angles

The 3D models for each lumbar vertebra D_i generated from layered CT scan images can be rotated to provide different views of the lumbar vertebrae. Matlab® incorporates two parameters that are azimuth angle and elevation angle which are used to define the orientation of a 3D object. With reference to the Matlab® documentation [16], the definitions of these angles are explained here. Let x be the axis in the right direction, y be the axis in the direction straight ahead going away and z be the axis in the up direction as depicted in Figure 3-3. Then, the azimuth angle is defined as the viewing angle in the xy plane with positive values indicating counter-clockwise rotation from the viewpoint and vice-versa. The elevation angle was defined as the viewing angle made above or below the xy plane, here positive values of elevation angle indicate that the angle was made above the xy plane and negative values of elevation angle indicate that the angle was made below the xy plane. An illustration of these angles is as shown in Figure 3-3. For this study, the orientation of lumbar vertebrae will be described using the pair of these two angles, (az_angle, el_angle), where az_angle represents the azimuth angle of viewing the object and the el_angle represents the elevation angle of the viewing the object. All angles mentioned in this study are measured in degrees, unless specified otherwise. The orientation corresponding to $(-90, 0)$ represents the front view of the object.

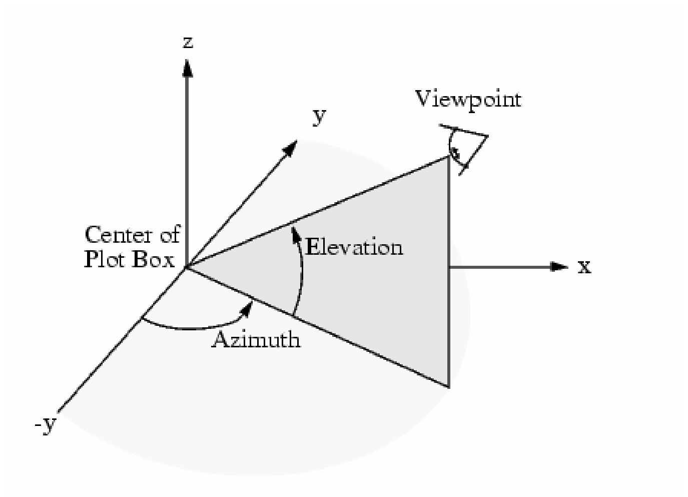

Figure 3-3: Azimuth and elevation angles [16].

3.1.3. Cropping the pedicle

As seen for 3D models for lumbar vertebrae L1 and L2 in Figure 3-1 and Figure 3-2, the set D consists of 3D models that represent the complete structure of lumbar vertebrae L1−L5. For this research undertaking, we required only the vertebrae without their pedicle portion that connects to the vertebral column. Hence, we revert back to the algorithms explained in sub-section 3.1.1. The given algorithms were modified so as to cut out the pedicle portion from each of the 3D models, D_i.

To cut out the pedicle portion, we first needed to compute the top view of the vertebra in order to compute the position from where the pedicle portion of the vertebra begins to project out. The smoothened three-dimensional structure, J_s which was obtained by stacking all the binary images in the set B, represents the structure of the lumbar vertebra model as described in sub-section 3.1.1. In order to proceed with the cropping of the pedicle, we need to find the top view image of the three-dimensional structure J_s. To compute the top view projection of J_s, a logical AND operation was performed over all the binary images b, $b \in B$, that make up the structure of J_s. The top view image obtained for L1 is shown in Figure 3-4. In the top view image, we compute the location from where the pedicle is attached to the rest of the vertebra. Using this location we crop the pedicle out from the remaining structure of the vertebra so as to retain only the vertebra part.

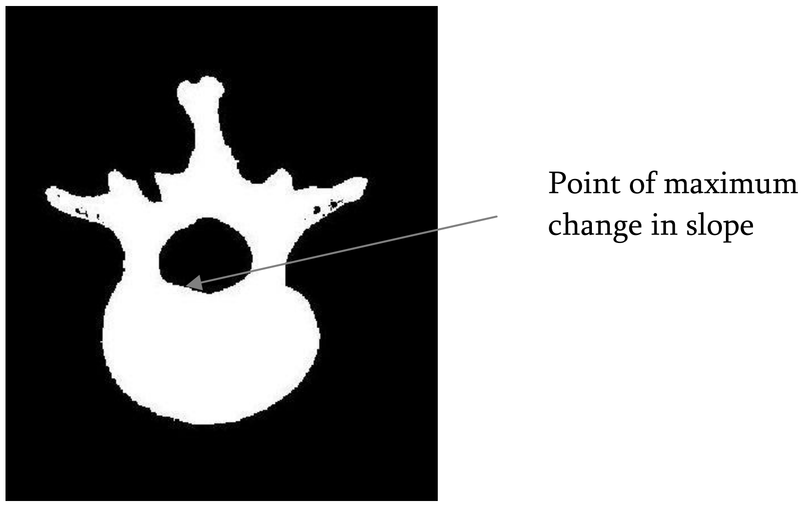

Figure 3-4: Top view of 3D model L1.

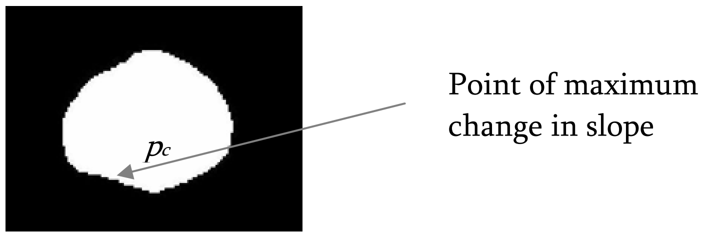

Figure 3-5: Center blob from top view of 3D model for L1.

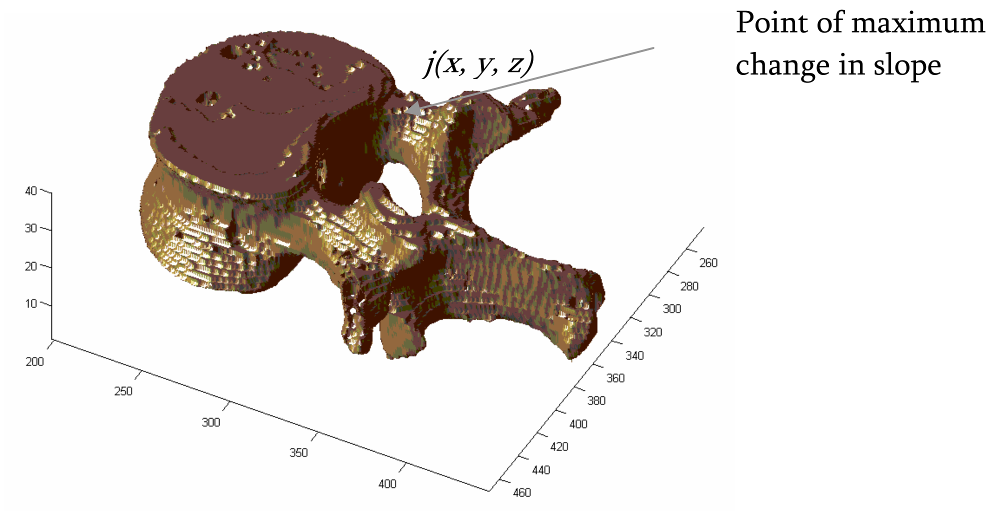

Figure 3-6: 3D model L1 showing optimal pixel to cut off pedicle.

The center dark blob in the top view image in Figure 3-4 was segmented out as shown in Figure 3-5 as a white blob. For the segmented center blob, a set of points B_b are determined, such that every point p in B_b lies on the boundary of the bottom half of the center blob. From the set of point B_b, a point p_c is found such that the change of slope for the curve defined by the points in B_b is the greatest. This point p_c denotes the optimum position from where the pedicle part should be separated out from the rest of the vertebra. This point p_c was then transformed to its equivalent position $j(x, y, z)$, $j \in J_s$, where J_s is the three dimensional structure of the lumbar vertebra described in sub-section 3.1.1. The point j is illustrated in Figure 3-6. All the pixels beyond the vertical orthogonal plane parallel to the yz plane and passing through the point j are then changed to background pixels, hence eliminating the pedicle part from the vertebra. Let this structure be called J_c. Next, the *patch* routine available in Matlab was then applied to the vertebra structure without the pedicle, J_c, to generate the required 3D model. This process was repeated for all the complete 3D models in set D to generate the required 3D models D_c. Figure 3-7 through Figure 3-11 show the 3D lumbar vertebrae models generated after cropping the pedicle.

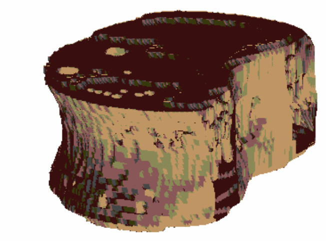

Figure 3-7: 3D model for lumbar vertebra L1 after cropping out the pedicle.

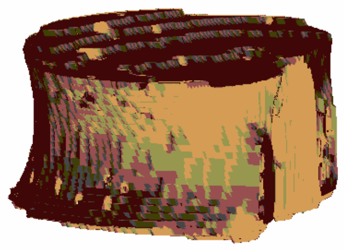

Figure 3-8: 3D model for lumbar vertebra L2 after cropping out the pedicle.

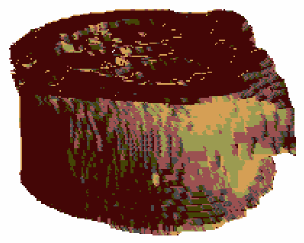

Figure 3-9: 3D model for lumbar vertebra L3 after cropping out the pedicle.

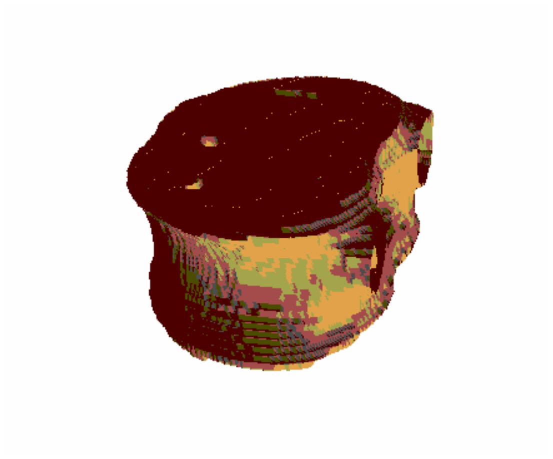

Figure 3-10: 3D model for lumbar vertebra L4 after cropping out the pedicle.

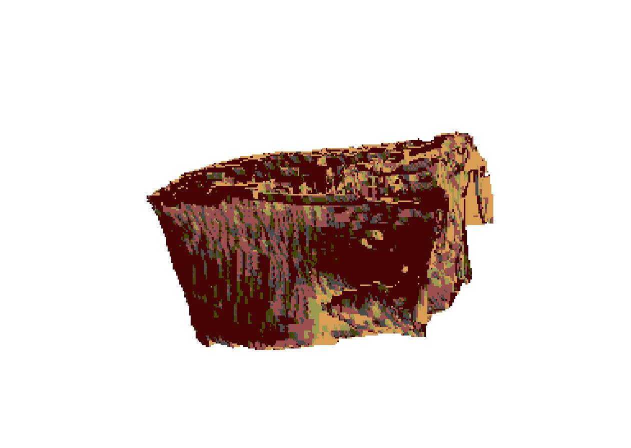

Figure 3-11: 3D model for lumbar vertebra L5 after cropping out the pedicle.

3.2. CREATING AND SAVING THE PROJECTION IMAGES

3.2.1. Projection of 3D models

In an attempt to employ 3D models of lumbar vertebrae in discriminating the presence of traction in lumbar vertebrae, we need to compare the structure of 3D models to the structure of vertebrae in our data set. In this research, we compute the orientation of the 3D models at different angles and find the best matching orientation of the 3D models to the two dimensional vertebrae boundaries found for vertebrae in the x-ray images. The lumbar vertebrae boundaries were determined from their respective files provided by NLM, containing 36 points along the vertebra boundary, using the same procedure as presented in section 2.1.1. The lumbar vertebrae from x-ray images were then matched to the corresponding 3D models of lumbar vertebrae of the same type, L1–L5.

The lumbar vertebrae models generated are three dimensional in structure, the data set of the vertebrae to be discriminated for traction can be provided in form of x-rays or other two dimensional image forms. Hence, in order to compare the provided vertebrae with the 3D models, it was required to find the two dimensional projection of the 3D models at different combination for the angles of orientation. The different orientation angles used in this study are between -100 and -80 for azimuth angle and -10 and 10 for elevation angle with an interval of 0.5 for each. The range of different values for the azimuth and elevation angles is given by $azRange$ and $elRange$ respectively.

$$azRange = \{-100 + (0.5)t \mid 0 \le t \le 40\} \tag{3.2}$$

$$elRange = \{-10 + (0.5)t \mid 0 \le t \le 40\} \tag{3.3}$$

The sets $azRange$ and $elRange$ both contain 41 different values. For each $az\_angle \in azRange$ and $el\_angle \in elRange$, the projections of a 3D model corresponding to a combination pair of (az_angle, el_angle) gives a total of $1{,}681$ different projection of the 3D model.

3.2.2. Storing and indexing of projection images

In order to compute the orientation characteristics of lumbar vertebrae in the data set provided, it was required to generate the projections of a corresponding 3D model for all $1{,}681$ different combinations of (az_angle, el_angle) to find the best match. The process of computing the projections of a 3D model for all different angles had to be performed for each case of lumbar vertebrae in the experimental data set. The process of generating the projection for a 3D model was time consuming and was recurring for each experimental case. Hence, it was decided to compute all the different projections for each of the 3D models D_{c_i}, store them and retrieve them whenever needed. For each 3D model D_{c_i}, we obtain a set P_i, which contained the $1{,}681$ different projections of that lumbar vertebra model. Next, it was required to save the all the projection images obtained for all the five models for lumbar vertebrae L1−L5, and to index them for easy retrieval. The indexing of the projection images involves creating a index for each projection images based on, first, the label indicating the type of lumbar vertebra viz. L1−L5 and second, the pair of angles (az_angle, el_angle) which determined the orientation of the corresponding 3D model for which the projection was obtained. The filenames used for projection images contained a label indicating the type of lumbar vertebra and the corresponding (az_angle, el_angle) angles of the projection. Hence, the filenames of the projection images themselves act as an index, which can be used for indexing of these projection images.

While indexing and storing the projection images, the aspect ratio of the projected vertebra model was also calculated. The comparison of projection images of the vertebra models with the lumbar vertebra images in the data set needed to be performed to compute its orientation characteristics. It was required to calculate the aspect ratios of the projection images as the resolution of the projection images and the resolution of the vertebra images in the experimental data set could differ non-linearly.

For all the vertebrae provided in the data set, it was seen that the resolution of the vertebrae images in the data was much higher than that of the projection images of 3D models. Hence, in order to make the projection images comparable in size to that of the vertebrae in the dataset, it was required to resize the projection images for each vertebra in the dataset. The length of the posterior side and the length of the superior side are chosen as representative of the aspect ratios of the projection of the 3D models of lumbar vertebrae. The length of the posterior side of the lumbar vertebra in the projection image $postDist3D$ and the length of the superior side $topDist3D$ are calculated for each of the projection images, P_i, for all the five 3D models. In order to calculate $postDist3D$ and $topDist3D$, first the corner pixels of the projection of the 3D vertebra model in the images P_i are computed. Using, the corner pixels, the lengths of the sides can be calculated, Figure 3-12 illustrates the lengths of the posterior and superior sides that are calculated. Also, using the corner pixels the slope of the posterior side $slopePost3D$ for the projected vertebra P_i was computed.

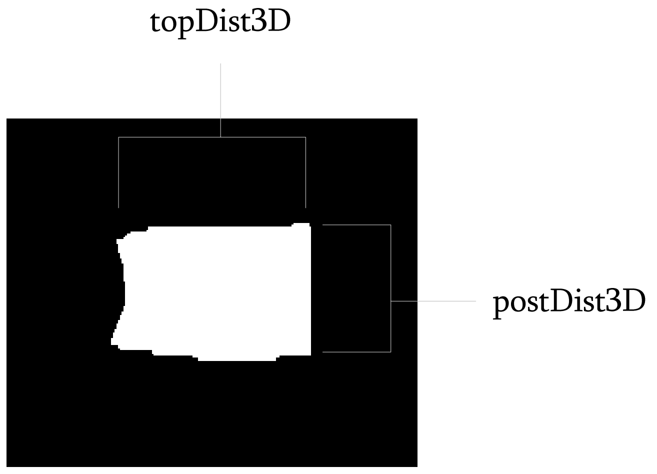

Figure 3-12: A projection of lumbar vertebra L1 at viewing angles (-90.0, 1.0). Lengths of posterior and superior sides are labeled.

3.3. ALGORITHM TO COMPUTE ORIENTATION CHARACTERISTICS OF LUMBAR VERTEBRAE

3.3.1. Determination of vertebral boundary

For each lumbar vertebra in the data set we are provided with a text file which consists of (x, y) pairs of coordinates of 36 points along the boundary of the vertebra. These 36 points are marked along vertebra boundary by experienced radiologists and domain experts. In order to determine the orientation characteristics of the lumbar vertebra by comparison to the 3D models, we need to compute the shape of the lumbar vertebra. A second order B−spline [2] algorithm was applied to the set of 36 coordinates which computes a set of connected points that make up the complete vertebra boundary. An image fill operation was performed upon the set of connected boundary pixels to get the completely filled vertebra as shown in Figure 3-13. If $L_f = L_f(x, y)$ denotes the filled vertebra, then L_f was defined as,

$$L_f(x, y) = \begin{cases} 1, & \text{if } (x, y) \text{ lies inside or on the vertebra boundary} \\ 0, & \text{elsewhere.} \end{cases} \tag{3.4}$$

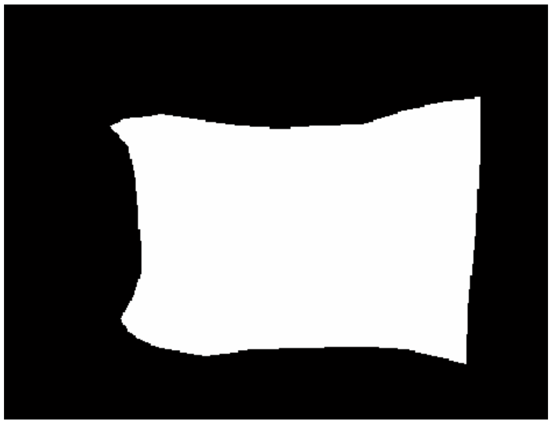

Figure 3-13: A filled lumbar vertebra obtained by B-spline and image fill operation over the 36 boundary points.

3.3.2. A customized algorithm for resizing images

Each lumbar vertebra L_f needs to be compared with all the projections P_i of its corresponding 3D model in order to compute the best matching 3D projection for that lumbar vertebra to determine its orientation characteristics. The resolution of each lumbar vertebra and the resolution of the projections P_i of the 3D models were expected to vary non-linearly. It was seen that the resolution of lumbar vertebrae in the provided data set was much larger compared to the resolution of the projection images. Hence, the projection images are to be resized by up–sampling. Also, the aspect ratios of projections of 3D models and of the lumbar vertebrae are different, that is, the size of projections along the posterior edge and the size of the projections along the superior edge vary by different factors to that of the lumbar vertebrae images. Therefore, projection images are to be resized by a factor M along one dimension and a different factor N along the other dimension. The resize algorithm provided in Matlab® was customized so as to resize the rows of the input image by a factor M and the columns of the input image by a factor N. This was done by first resizing the projection image only for the row dimension by a factor M, and then next, the resulting image was then resized only for the column dimension by a factor N to produce the resized projection image. Hence, the resizing of the projection image P_i using different resizing factors M and N could be performed to get the resized projection image P_{R_1}.

For each lumbar vertebra L_f, the resize operation was to be performed for each projection P_i. The resizing operation was the dominating factor in determining the computation time of the entire process of determining the orientation characteristics of the vertebra L_f. Hence, an attempt was made to optimize the resizing operation in order to reduce its computation time. Since, the images to be resized are binary images, the resizing of projection images was performed using only the boundary points of the object in the image. This resizing operation was performed using different resizing factors along the row and column dimension. The optimization in resizing images is explained by the following procedure. First, for the projection image P_i, let M and N be the row and column scaling factors, respectively, between the projection image P_i and vertebra image. Second, boundary extraction [17] was performed on the projection image P_i so as to produce P_t, the boundary of the projection in the image with thickness of one pixel. The boundary extraction can expressed as a set difference operation given by Gonzalez et al. [17] as,

$$P_t = P_i - erode\left(P_i, \hat{B}\right) \tag{3.5}$$

where, $erode\left(P_i, \hat{B}\right)$ refers to the erosion operation of P_i with $\hat{B}$, and $\hat{B}$ refers to the structuring element taken as,

$$\hat{B} = \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \tag{3.6}$$

Third, the customized resize algorithm was applied to P_t using M and N as resizing factors to produce a resized boundary of the projection P_{R_t}. Fourth, an image fill operation was performed on P_{R_t} to generate the resized projection of the 3D model, P_{R_2}. P_{R_1} and P_{R_2} refer to the same resized projection image, but are computed differently. P_{R_1} was computed by applying the customized resize algorithm with resizing factors M and N, while P_{R_2} was computed by extracting the boundary of the projection P_i, next applying the resizing operation with resizing factors M and N, and lastly performing an image fill to produce P_{R_2}. The computation time for both the procedure were recorded over a randomly chosen test set of P_i as shown in Table 2-1. The results show that the second method of resizing projection images has better computation time. The results show that the second method of resizing images consistently provided an approximate reduction in computation time by 40% on an average for the resize operation over the randomly chosen projection images. The resizing operation directly depends upon the number of points in an image to be resized. Hence, the second method performed faster as the number of points in the projection image was reduced by only considering the points at the boundary of the projection.

Table 3-1: Computation times of resizing images for the investigated resizing functions.

Resizing factors (M, N) used to resize P_i	Time to compute P_{R_1} (sec.)	Time to compute P_{R_2} (sec.)
(2.389, 1.734)	0.235	0.125
(2.528, 1.734)	0.234	0.156
(2.583, 2.037)	0.282	0.141
(3.35, 2.064)	0.281	0.203
(5.024, 3.169)	0.437	0.297
(5.452, 2.843)	0.390	0.265
(2.478, 1.536)	0.250	0.156
(2.691, 1.956)	0.234	0.172

3.3.3. Algorithm to compute the best matching 3D projection for each lumbar vertebra

In order to compute the best matching 3D projection for each lumbar vertebra, the operations to be performed on the images can take large computation time. Since most of the operations to be performed on images directly depend on the number of pixels in the image, the images are thus cropped to optimize the computation time. The minimum required resolution was calculated to be $imgRows \times imgCols$ and was set as the resolution for all the images required during computation of intermediate and final results for the given experimental case. The filled vertebra $L_f(x, y)$ was cropped to the size $imgRows \times imgCols$ to get the set of points $L(x, y)$, where $1 \le x \le imgRows$ and $1 \le y \le imgCols$.

The aspect ratios of the vertebra L are calculated. For this the coordinates the off the corner pixels of the vertebra L are computed. Using the coordinates of the corner pixels, the length of the posterior side $postDist$ and the length of the superior side $topDist$ are calculated. Using the coordinates of the corner pixels of the posterior side, the slope of the posterior side $slopePost$ of vertebra was also calculated. The value of $slopePost$ was used to rotate the vertebra defined by the set of points in L such that the posterior side of the resulting vertebra was exactly vertical. The resulting vertebra was then translated (shifted) such that the centroid $(\bar{x}_L, \bar{y}_L)$ was positioned at the center of the image which corresponds to the position $\left(\frac{imgRows}{2}, \frac{imgCols}{2}\right)$. Let the set of points in the translated lumbar vertebra be denoted by L_R.

$$L_R(x,y) = \begin{cases} 1, & \text{if } (x,y) \text{ is a point on the vertebra} \\ 0, & \text{elsewhere.} \end{cases} \tag{3.7}$$

where, $1 \le x \le imgRows$, $1 \le y \le imgCols$, and the centroid of L_R was given by,

$$(\bar{x}_L, \bar{y}_L) = \left(\frac{imgRows}{2}, \frac{imgCols}{2}\right) \tag{3.8}$$

The following process explains the process of computing the best matching projection of 3D models for the vertebra L_R. The vertebra L_R was to be compared with the projections of all the combinations of azimuth and elevation angles (az_angle, el_angle) of the 3D vertebra model D_{c_i} corresponding to the type of vertebra L1–L5. For each combination of (az_angle, el_angle), we retrieve the projection image P_i of 3D model D_{c_i} and the following procedure was carried out. First, the aspect ratios, $postDist3D$ and $topDist3D$, and the slope of the posterior side, $slopePost3D$, for the projection P_i were retrieved, as explained in sub-section 3.2.2. Second, using the value of $slopePost3D$, the projection image P_i was rotated such that the posterior side of the projection becomes exactly vertical generating a rotated form of the projection P_θ. Note that P_θ was just the rotated form of P_i, hence the aspect ratios and other characteristics of P_i and P_θ remain the same. Third, the resizing factors were calculated using the values $postDist3D$ and $topDist3D$ for the projection P_θ and the values $postDist$ and $topDist$ for the lumbar vertebra L_R. The resizing of the projection P_θ was necessary because the resolution of P_θ and L_R can differ. All of the vertebra cases explored in our experimental data set showed that the resolution of L_R was much higher than that of P_θ. Therefore, resizing of P_θ was performed by up−sampling to make the resolution of P_θ suitable for comparing with vertebra L_R. For all the projections for different combinations of (az_angle, el_angle) the resolutions differed non-linearly. Accordingly, the resizing of projection images was performed at run−time for each vertebra case L_R to be studied. The resizing factors calculated are,

$$M = \frac{postDist}{postDist3D} \tag{3.9}$$

$$N = \frac{topDist}{topDist3D} \tag{3.10}$$

Fourth, the projection P_θ was resized using the customized resizing operation explained in sub-section 3.3.2 to produce the resized projection image. The set of points in the resized projection image are then translated such that its centroid $(\bar{x}_P, \bar{y}_P)$ lies at the center of the image, then its image size was reduced to $imgRows \times imgCols$ by cropping out background pixels from the image boundaries. Let the resized projection image be P_R. The centroid for P_R was $(\bar{x}_P, \bar{y}_P) = \left(\frac{imgRows}{2}, \frac{imgCols}{2}\right)$. Fifth, the projection image P_R and the vertebra L_R were compared by taking an exclusive−OR, defined as,

$$X(x,y) = L_R(x,y) \oplus P_R(x,y) \tag{3.11}$$

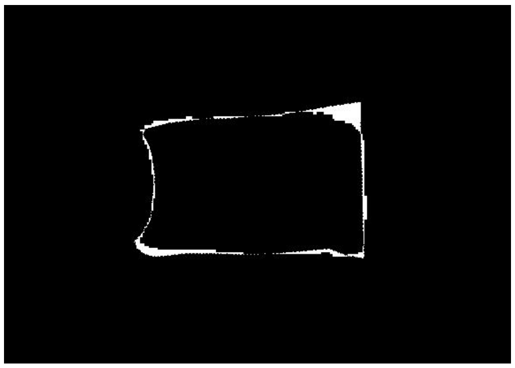

Figure 3-14: Exclusive-OR between lumbar vertebra L_R and its optimal projection.

Figure 3-14 shows the exclusive−OR obtained for the lumbar vertebra in Figure 3-13 of type L1 and its best matching projection of the cropped 3D model D_{c_1}. The exclusive−OR was representative of the comparison between the lumbar vertebra L_R and the 3D projection, P_R, at a particular combination of viewing angles (az_angle, el_angle). Sixth, the area of region described by the set points in X was calculated and is denoted by A_X. The above procedure was repeated for each combination of $(az\_angle,\ el\_angle)$ for a lumbar vertebra L_f. Hence, we get a set of values corresponding to A_X for each projection corresponding to a combination pair of $(az\_angle,\ el\_angle)$. A table *xorAreas* was maintained that maps each combination $(az\_angle,\ el\_angle)$ of the projections to the area of the exclusive−OR image, A_X computed for those viewing angles.

$$xorAreas(az\_angle, el\_angle) = A_X \tag{3.12}$$

Now, the best matching projection P_{op} of the 3D model, $P_{op} \in \{ P_i \}$, for the vertebra L_f was determined based on which projection P_i of the 3D model had the least exclusive−OR area found from the table $xorAreas$. The viewing angles of the 3D model corresponding to the best matching projection describe the orientation characteristics of the lumbar vertebra L_f. Let X_{op} denote the exclusive−OR for best matching projection P_{op}, $(azOptimal,\ elOptimal)$ be the viewing angles of the 3D model corresponding to the projection P_{op} and the area of the exclusive−OR X_{op} can found as, $A_{X\,op} = xorAreas(azOptimal, elOptimal)$. Thus, the orientation characteristics for the lumbar vertebra L_f are computed and, X_{op}, $(azOptimal,\ elOptimal)$ and $A_{X\,op}$ are computed for each lumbar vertebra L_f and are saved. The following algorithm summarizes the process of calculating orientation features for lumbar vertebrae.

Inputs:

Shape36Filename	*The text file containing the coordinates of 36 points on the boundary of the vertebra.*
ModelInfoFilename	*The file where aspect ratios of projections are stored.*

Outputs:

azOptimal	*The azimuth angle of orientation for the vertebra of the given case.*
elOptimal	*The elevation angle of orientation for the vertebra of the given case.*

<u>Algorithm:</u>

```
1. $L_{in} \leftarrow$ Read the 36 coordinates of vertebra boundary points from the file
   Shape36Filename.
2. $L_b \leftarrow B - Spline(L_{in})$
3. $L_f \leftarrow Image\_fill(L_b)$
4. Calculate $(imgRows, imgCols)$, which are the minimum dimensions of the image
   required to represent the given vertebra $L_f$.
5. $(top, left, right, bottom) \leftarrow$ Calculate corner pixels of the of vertebra $L_f$.
6. Calculate length of posterior side and length of superiorside,
       $postDist = \sqrt{(top.x - right.x)^2 + (top.y - right.y)^2}$
       $topDist = \sqrt{(top.x - left.x)^2 + (top.y - left.y)^2}$
7. Calculate the slope of the posterior side of vertebra $L_f$,
       $slopePost = \frac{(top.y - right.y)}{(top.x - right.x)}$
8. Compute $L$ with the following steps; using the value of $slopePost$ rotate $L_f$ such
   that the posterior side was vertical, crop the image to $imgRows \times imgCols$, next
   translate (shift) the points in the vertebra image such that the centroid lies at
   $\left(\frac{imgRows}{2}, \frac{imgCols}{2}\right)$.
9. $index$, is the value indicating the type of lumbar vertebra L1−L5 for the given case.
10. $azRange = \{-100 + (0.5)t \mid 0 \le t \le 40\}$
11. $elRange = \{-10 + (0.5)t \mid 0 \le t \le 40\}$
12. For each $az\_angle \in azRange$
    12.1. For each $el\_angle \in elRange$
        12.1.1. Retrieve projection $P_i$ for the angles $(az\_angle, el\_angle)$ for the 3D
                model corresponding to $index$.
        12.1.2. Compute $P_\theta$, by rotating $P_i$ such that posterior side of the projection was
                vertical.
        12.1.3. Retrieve aspect ratios $postDist3D$ and $topDist3D$, from the file $ModelInfoFilename$ corresponding to $(az\_angle,\ el\_angle)$ and $index$.
        12.1.4. Resizing factors,
                $M \leftarrow \frac{postDist}{postDist3D}$; $N \leftarrow \frac{topDist}{topDist3D}$
        12.1.5. $P_R \leftarrow Resize(P_\theta, M, N)$
        12.1.6. Crop $P_R$ to size $imgRows \times imgCols$ and translate the points in the projection such that the centroid of the projection lies at $\left(\frac{imgRows}{2}, \frac{imgCols}{2}\right)$.
        12.1.7. Compute exclusive−OR $X$ between lumbar vertebra and current projection,
                $X(x,y) = L_R(x,y) \oplus P_R(x,y)$
        12.1.8. $A_X \leftarrow Area(X)$
        12.1.9. $xorAreas(az\_angle,\ el\_angle) \leftarrow A_X$
13. $minArea \leftarrow minimum(xorAreas)$
14. Find the values, $azOptimal \in azRange$ and $elOptimal \in elRange$ such that $xorAreas(azOptimal, elOptimal) = minArea$.
15. Save $X$ and $P_R$ corresponding to $(azOptimal, elOptimal)$.
```

3.4. EXPERIMENTS PERFORMED

3.4.1. Experimental Data

The experimental data was provided by the National Library of Medicine (NLM), which contained the following:

1) A data sheet consisting of a table where each row was a tuple τ, $\tau = (name, c_S, c_I, t_S, t_I, o_S, o_I)$. Here, the attribute *name* contained a string for the vertebra name. The attributes c_I and c_S have values *true/false* indicating the presence of claw on the superior and inferior sides of the vertebra respectively. The attributes t_S and t_I have values *true/false* indicating the presence of traction on the superior and inferior sides of the vertebra respectively. Lastly, the attributes o_S and o_I have enumerated labels $\{slight, moderate, severe\}$ indicating a grade for the presence of anterior osteophytes on the superior and inferior sides of the vertebra.
2) For each vertebra in the data sheet, a text file was provided which contained values representing (x, y) coordinates of 36 points along the vertebral boundary for the corresponding vertebra.

This study aims at discriminating lumbar vertebrae for the presence of traction and hence, only the truth labels for the presence of traction are required and so, the tuples in the data sheet are reduced to $\tau = (name,\ t_S,\ t_I)$. The data set provided consisted of a total of 261 lumbar vertebrae for which the proposed orientation characteristics were calculated in order to facilitate in determining the presence of traction in the lumbar vertebrae. The 36 points along the vertebral boundary for each vertebra were provided to NLM by experienced radiologists and domain experts. For the entire dataset, a new class of attributes t was introduced, which had values labeled *true/false*, where t was indicative of the presence of traction for that lumbar vertebra. The attribute class, t, indicative of the presence of traction was assigned a value *true*, if either t_s, the attribute class for presence of traction at superior side or t_I, the attribute class for presence of traction at the inferior side had a value $true$; otherwise it was assigned the value $false$.

The data set was stratified by the type of lumbar vertebrae, which was L1 – L5. It was observed that the data set of 261 lumbar vertebrae consisted of 12 L1s, 42 L2s, 75 L3s, 78 L4s and 54 L5s. The 261 entries in the data set when grouped by the target variables t showed the following distribution.

Table 3-2: Distribution of lumbar vertebrae cases based on type of lumbar vertebra.

Type of Lumbar vertebra	Number of Lumbar vertebrae (Traction \ No Traction)
L1	9/3
L2	21/21
L3	35/40
L4	27/51
L5	17/37

The features $F_1 - F_5$ as explained in [2, 3] are calculated for each lumbar vertebrae provided in the data set. The features $F_1 - F_5$ based on the convex hull of the vertebrae were developed in order to discriminate lumbar vertebra for the presence of traction. These are the same features as explained in section 2.1 The feature $F_1 - F_5$ were also calculated the optimal projection P_{op}, obtained for each lumbar vertebra in the data set. The optimal projections P_{op} describe the orientation characteristics of the lumbar vertebrae, as explained in sub-section 3.3.3. Let the features calculated for the optimal projections be denoted by $F_{1p} - F_{5p}$.

3.4.2. Training data and test data

In order to generate the training set, we integrate the features $F_{1p} - F_{5p}$ calculated for the optimal projection P_{op} for each lumbar vertebra in the data set and the optimal orientation characteristics with the data provided by NLM. Hence, the integrated data set obtained consisted of a set of tuples of the form, $\tau_m = (name, F_{1p}, F_{2p}, F_{3p}, F_{4p}, F_{5p}, azOptimal, elOptimal)$, where, $azOptimal$ and $elOptimal$ are the orientation characteristics and $F_{1p} - F_{5p}$ correspond to the features calculated for the optimal projection P_{op} for the lumbar vertebra in the data set corresponding to $name$.

To generate the test data set, we integrate the features $F_1 - F_5$ calculated for the lumbar vertebrae in the data set with the optimal orientation characteristics computed for each lumbar vertebra and the truth labels indicating the presence of the traction as provided in the data by NLM. Hence, the test data set consisted of set of tuples of the form, $\tau_n = (name, F_1, F_2, F_3, F_4, F_5, azOptimal, elOptimal, t)$; where t was the label indicating the presence of traction, as explained in sub-section 3.4.1 and $F_1 - F_5$ are the features calculated for the lumbar vertebra corresponding to $name$.

3.4.3. Classification

The classification problem here involved generating a model that can classify a given case of lumbar vertebra for the presence of traction into classes traction/no traction. To generate the trained model for each model L1 – L5 the following procedure was applied to the corresponding training sets. First, for each feature F_p in the set of features $F_{1p} - F_{5p}$ of the training data, the mean and the standard deviation values, μ and σ were calculated for all features F_p in the feature set $F_1 - F_5$ of the training set.

Second, the features are normalized by subtracting its mean from each feature and dividing by its standard deviation. For each feature F_p in the feature set, we calculate the standardized feature F_{Pnorm} as, $F_{Pnorm} = \{f_{norm} \mid \forall f \in F_p, \exists f_{norm}\}$, where f_{norm} was calculated as, $f_{norm} = \frac{f - \mu}{\sigma}$, for $f_{norm} \in F_{Pnorm}$ and $f \in F_p$. Third, the number of clusters for the class no traction was found using subtractive clustering [12, 13]. All the features F_p were calculated based on the projection images of 3D models which are representative of a normal lumbar vertebra without any traction. Hence, all the tuples in F_p belong to the class no traction. Fourth, using the normalized featured vectors for the training data and the number of clusters estimated for each class, K-means clustering [14,15] was performed to determine the cluster centers for the class no traction. The cluster centers for the training set corresponding to each lumbar vertebra L1 – L5 are saved along with their corresponding mean and standard deviation values of each feature.

For each of the feature vectors in the test set corresponding to each lumbar vertebra L1 – L5, nearest centroid classification was performed. First, each feature F in the test set $F_1 - F_5$ was standardized using z-score score normalization. For each feature F in the feature set $F_1 - F_5$, using the mean and standard deviation values calculated and from the training set features, we calculate the normalized feature F_{norm} as, $F_{norm} = \{f_{norm} \mid \forall f \in F, \exists f_{norm}\}$, where f_{norm} was calculated as, $f_{norm} = \frac{f - \mu}{\sigma}$, for $f_{norm} \in F_{Pnorm}$ and $f \in F_p$. Second, the cluster centers calculated and saved for the training set are retrieved. Third, for each normalized feature vector, $minDist$ the minimum of the Euclidean distance to each of the cluster centers was calculated. Fourth, the ordered pair of $(name, minDist)$ was latched to either the list $abnormalList$ or $normalList$ depending upon the label t indicating the presence or absence of traction for this feature vector. Fifth, the two lists of Euclidean distances are input to a routine which computes the area under the Receiver Operating Characteristics (ROC) curve [18] based on the true positive and true negative cases obtained for the test set. True positive refers to the test case vertebrae with traction being classified correctly and true negative refers to the test case vertebrae with traction being classified incorrectly. Sixth, the area under the ROC curve is recorded. This process is repeated for all the five lumbar vertebrae L1 – L5 providing the value of the area under the respective ROC curves.

3.5. RESULTS AND DISCUSSION

3.5.1. Experimental Results

The results obtained from the experiments performed in section 3.4 to classify lumbar vertebrae as traction/no traction using the convex hull based size invariant features is discussed below. The experiments were performed over the provided data set of 261 lumbar vertebrae whose distribution is provided in Table 3-2. The classification was performed by generating a training model using K-means clustering over the size-invariant features computed for the best matching projection of each lumbar vertebra in the data set. The test set was generated by computing the size-invariant features using the lumbar vertebra x-ray images for each lumbar vertebra in the data set. Hence, the training set consisted of 261 tuples generated using the best matching projection of the provided lumbar vertebrae and the test set consisted of 261 tuples generated using the x-ray images of the lumbar vertebrae.

Table 3-3: Areas obtained under Receiver Operating Characteristic (ROC) curve for classifying lumbar vertebra L1-L5 for traction, respectively.

Type of Lumbar vertebra	Area under Receiver Operating Characteristic (ROC) curve
L1	0.89
L2	0.68
L3	0.60
L4	0.53
L5	0.60

3.5.2. Conclusions

In this research, methods were investigated and implemented to model lumbar vertebra in three dimensional structures. Several conclusions can be drawn from the approaches adopted in this research undertaking. First, computed tomography (CT) scan images were used to generate three dimensional models of lumbar vertebrae. The cross sectional CT scan images were used to produce layered slices of the lumbar vertebrae obtained by segmentation which could generate the 3D models. Hence, CT scan images could be used to generate 3D models of lumbar vertebrae successfully in order to study their shapes and structures.

Second, in order to compare the 3D models of lumbar vertebrae with 2D images from x-ray images of lumbar vertebrae, the projections of 3D models were computed at different viewing angles. This enabled comparing the 3D models with the x-ray image vertebrae. Indexing and storing of these projection images was performed, so that the computation times for each x-ray image were improved in overall calculation of orientation characteristics of each case.

Third, a 3D model was generated from the CT scans provided for a particular case where the lumbar vertebrae did not show any presence of traction. Hence, the 2D projection images of the 3D models for the lumbar vertebrae L1 – L5 were used to represent normal vertebrae. The shape and size invariant features calculated over the optimal projection images of each case were used to represent the characteristics of normal vertebrae which were input to a K-Means clustering algorithms to provide clustering-based models for vertebrae L1 – L5 to represent normal vertebrae. The shape and size invariant features calculated for the images obtained from x-ray images were used for testing. The experimental results did not show that the projections of the 3D model used provided features that were capable of distinguishing normal lumbar vertebrae from cases where traction was present for each type of lumbar vertebrae L1 – L5 vertebrae. The classification results for L1 were more encouraging than for the other lumbar vertebra cases for L2 – L5. However, there were only 12 lumbar vertebra x-ray images in the provided data set corresponding to L1 to support the accuracy of the model in predicting the presence of traction in lumbar vertebrae.

Fourth, it was observed that the 3D models and the resulting projection images had superior and inferior sides of the vertebra with convex edges. A majority of the x-ray images of lumbar vertebra showed superior and inferior sides having more flat like or concave edges. One of the major difficulties in generating the 2D projections representative of a vertebra based on the 3D CT scan-based model and the x-ray image vertebra is the resolution disparity. The CT scan images and the resulting projections of 3D models were much smaller in size and varied non-linearly in row and column aspect with the corresponding x-ray image vertebra. Several variations of resizing functions have been investigated for generating the 2D projections of the 3D models of the vertebra having similar dimensions to that of the x-ray image vertebrae for comparison using the exclusive–OR approach for orientation determination.

Fifth, another consideration or limitation in the experimental results presented is the relative limited data set for each vertebra in generating clustering models to represent them. The shapes of the lumbar vertebrae L1 – L5 differ. Model generation and clustering analysis for each case was considered separately. The distribution of the lumbar vertebrae in the provided data set is given by Table 3-2. It can be observed that the provided data set for each case is limited and therefore the clustering models generated for each are relatively inefficient in classifying lumbar vertebrae for presence of traction.

APPENDIX A

Read-Me file for the project 'Discrimination of Cervical Vertebrae for presence of Claw, Traction and Anterior Osteophytes'

A.1. MATLAB to C++ conversion

The document describes the procedure which allows calling MATLAB routines from C/C++. The method used here was to create a wrapper function around MATLAB routines and then creating Dynamic Linked Libraries for it to be used in C/C++.

We divided the procedure into two major steps:

1. Creating the Dynamic Linked Libraries for using the MATLAB routines.
2. Creating the workspace in C/C++ developer environment with the libraries included.

A.1.1. Creating the Dynamic Linked Libraries for using the MATLAB routines:

The Dynamic Linked Libraries are created through the 'mcc' command in the MATLAB compiler. The syntax of the command used to create the DLL files was:

```
mcc -W lib:<lib_name> -T link:lib <file1>  <file2> … <fileN>
```

where,

- The option '-W lib:<string>' creates wrapper functions for each .m file into a library.
- file1, file2,…,fileN are names of the .m MATLAB files stored in the same directory. These .m files are supposed to define the MATLAB routines which we want to call from C/C++.
- 'lib_name' was the name of the library that we wish to create.
- The option '-T link:lib' specifies the target to be a library file.

After executing the above command several files are generated and stored in the current directory of the MATLAB compiler. The description of these files was given below:

1. <u>C/C++ Header and Source code files:</u>

A wrapper C source file (here <file_name>.c) which contains a function of the library providing the C interface to each of the files <file1.m>, <file2.m>….<fileN.m>. A header (here <file_name>.h) was also generated which contains the prototype for each of the export function defined in the wrapper C source file. This header file must be included in all applications that need to these exported functions. Another C source file <file_name>_mcc_component_data.c was generated which includes all necessary information about path and initializations that are need by the MATLAB compiler or the MCR to use the library.

2. Module definition file:

A module definition file (.def) was created to provide all the information about the export functions. This file was used to link to the library.

3. Component Technology File (CTF):

A Component Technology File (CTF) file was an archive of all MATLAB related files (M-files) that are encrypted and together provided a deployable package.

4. Dynamic Link Library (.dll) file:

This was the shared library (binary) that was created. In this example a file with name <file_name>.dll will be generated and was loaded each time the calling function makes a call to any of the routines defined in it. For Operating Systems other than Windows, a different kind of a shared library may be required.

Several others exports file are created along with the above files. All of the above 8 files that are generated are stored in the same directory which was the current working directory in MATLAB while running the 'mcc' command. As far as the process of MATLAB to C++ code conversion goes, it was just required to include these files in the workspace of C++.

A.1.2. Creating the workspace in C/C++ developer environment with the libraries included:

The first step here was to open a C/C++ developer environment like Microsoft Visual C++ and creating a new Console application project (workspace).

In this new project, we include all of the files generated by the ‘mcc’ command; it was a good idea to copy all of these files to the workspace directory of this project. The C++ code which was to be written here was the code which will provide the input parameters (if any) to the MATLAB routines to be called.

Consider a library ‘libcal’ generated for a M-File with function defined as :
function[o1 o2] = calculate(i1, i2, i3)

To call such a function defined in an M-file the following need to be done in the C++ program:

1) Include all libraries related to libcal, which were generated by the ‘mcc’ command.
2) Declare a variable in C++ for each of the input and output parameters, and also initialize or derive values for the input parameters.

   Example: double I1, I2, I3, O1, O2;
3) Now declare a variable pointer with ‘mxArray*’ for each of them, this was a datatype used to store array for passing to MATLAB.

   Example: mxArray *in1, *in2, *in3, *in4, *in5, *out1, *out2;
4) Allocate appropriate space for each of these mxArray pointers.

   Example: in1=mxCreateDoubleMatrix(1,1,mxREAL);

   out1=mxCreateDoubleMatrix(1,1, mxREAL);

   where, 1,1 -> signify the [row x column] dimensions of the array

   mxREAL -> specifies that values to be real numbers.
5) Copy contents of variables I1, I2, I3 into in1, in2, in3 which will be the input values to the MATLAB routine ‘calculate’.

   Example: memcpy(mxGetPr(in1), &I1, sizeof(double));

   where, memcpy() was a function in C++ which copies a block of memory from one memory location to another which was defined in the library string.h.

&I1, was the memory location of the input value I1.

sizeof(double), specifies the amount of bytes to be copied.

6) Now, call the function ‘libcalInitialize()’ (defined in the library) to start the MATLAB compiler or the MCR.

7) Call the function ‘mlfcalculate(2, &out1, &out2, in1, in2, in3)’, which also defined in the libcal library. This calls the ‘calculate’ function in the M-file and results are stored in out1, out2.

8) Note the difference in the definition of calculate function in M-file and its C++ counterpart ‘mlfcalculate’. The output parameters of calculate routine in M-file are pointers which appear in the parameter list of the ‘mlfcalculate’ function. The first parameter of the ‘mlfcalculate’ function specifies the number of parameters which represent the left hand side variables in the ‘calculate’ routine and the remaining are the right hand side variables.

9) Now, call the fuction ‘libcalTerminate (defined in the library) to close the MATLAB compiler or the MCR.

10) Now, the results from the MATLAB routine are stored in out1, out2 which are of data type mxArray*. So, we copy the contents of these variables into our C++ variables O1, O2 which are of type double.

    Example: memcpy(&O1, mxGetPr(out1), sizeof(double));

11) De-allocate space to all the mxArray pointers (and any other pointers also). Freeing space allocated to mxArray pointers was done by the call ‘mxDestroyArray(mxArry*);

    Example: mxDestroyArray(in1);

12) Now, compile and run the C++ project.

A.2. K-Means Classifier for detecting Claw

The K-Means Classifier for detecting Claw was a project which calculates features on single vertebra and then applies K-Means clustering technique to classify it as NORMAL or ABNORMAL for the presence or absence of claw.

The initial workspace that was provided included the following files:

- compute_convex_hull_features_36Points.m
- compute_convex_hull_features_6features.m
- mergepts.m
- moment_norm.m
- connectspline.m
- kmeans_2class_kNearest_subcluster.m
- computeVectorDistance2class.m

The 'compute_convex_hull_features_36Points.m' was MATLAB script which read the co-ordinates of vertebra points from .shp36 file. This was done over a large number of files stored in the local directory. For every set of such vertebra points, the 'compute_convex_hull_features_6features' function was called from its corresponding M-file 'compute_convex_hull_6features.m'. This function in all calculates six features and the image area of the vertebra and returns to the calling script. The features calculated for all the vertebrae are then stored in an Excel Spread Sheet (xls). The function defined in mergepts.m and connectspline.m are used in the compute_convex_hull_features_36Points.m and the function defined in moment_norm was used by the compute_convex_hull_6features.m.

The 'kmeans_2class_kNearest_subcluster.m' was a MATLAB script which read the features of the vertebrae from the Excel Spread Sheet. It also read variables from the 'Parameters_claw.mat' file (MAT-file) which stored the trained clusters for classification of vertebrae on the basis of presence or absence of Claw. It also included the mean and standard deviation values for all features. Each set of features (corresponding to one vertebra) are normalized using the mean and standard deviation values from the MAT-file. This normalized set of features along with the cluster centers for classifications are given as input to the function defined in 'computeVectorDistance2class.m'. In this function the actual classification takes place. With the normalized features it checks the distances of each feature to it

For the project KMeans Classifier for detecting claw, the above MATLAB code needed to be converted to C++. It was also required that the classification be done for one given vertebra at a time. The working of the code explained above worked on a large set of vertebrae images in Batch-mode. Hence, the two files that were changed from the above code were:

- compute_convex_hull_features_36Points.m
- kmeans_2class_kNearest_subcluster.m

The changes made to compute_convex_hull_features_36Points.m, so that it worked on a single vertebra were made and stored in the same file. The code of kmeans_2class_kNearest_subcluster.m was rewritten in C++ which worked upon a set of features for only one vertebra. This code then calls the computeVectorDistance2class function.

Also, in the provided workspace calculated the six features were calculated and stored in an Excel Spread Sheet. Then, the Kmeans classifier script was run for classifying by reading back the features from the Excel Spread Sheet. This project calculates the six features and then run the classification program on it, thus eliminating the use the Excel Spread Sheet.

A.2.1. Creating the Dynamic Linked Libraries for using the MATLAB routines:

All the files described above are kept in a single folder. In the MATLAB compiler, set the current working directory to the directory where all the files are stored and then, run the command:

```
mcc  -W  lib:libfcmc  -T  link:lib  compute_convex_hull_features_36Points
computeVectorDistance2class
```

After running this command, the files that are generated in the current directory are:

- libfcmc.c
- libfcmc.h
- libfcmc_mcc_component_data.c
- libfcmc.dll
- libfcmc.lib
- libfcmc.exports
- libfcmc.exp
- libfcmc.ctf

A.2.2. Creating the workspace in C/C++ developer environment with the libraries included:

In Microsoft Visual Studio .NET 2005, create a new Win32 Console project. Include the following files in the source code:

```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
#include"mat.h"
#include"libfcmc.h"
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libfcmc.lib")
```

The C++ implementation was divided into these parts:

1) Reading the MAT-file and extracting the cluster centers for classification and extracting the mean and standard deviation values for these features.

2) Reading the .shp36 files to get the vertebra points and then calling an appropriate MATLAB routine to calculate features.
3) Calling the MATLAB routine to classify for the presence or absence of claw based on the values of these features.

The file name of the .shp36 file was an input parameter taken as a command line argument. The filename to be passed here was to be the absolute path on the local machine or a relative path can be given if the .shp36 file in stored in the same or one the subfolders of the current workspace. Hence, the process can be worked on any .shp36 file just by passing different file names at different calls.

A class called ‘Data’ was created which stores all the values of features for a vertebra also it stores all the values extracted from the MAT file, which are the centers of the classification clusters and the mean and standard deviation values of the features. Several functions are defined that work on these member variables. Keeping a separate class for the features values and for the classification parameters was considered redundant as the program was just suppose to work on one given vertebra at a time.

Since, there are more than one function calls to the MATLAB routines, the initialization and termination of the MATLAB compiler are done close to the entry and exit points of the C++ program.

The call to the program was given on the command prompt as:

```
C:\<WorkSpaceDir>\KMeansClassifier_Claw C:\vertebra\C01235_3.shp36
```

A.3. K-Means Classifier for detecting Traction

The K-Means Classifier for detecting Traction was a project which calculates features on single vertebra and then applies K-Means clustering technique to classify it as NORMAL or ABNORMAL for the presence or absence of traction.

The initial workspace that was provided included the following files:

- compute_convex_hull_features_36Points.m
- compute_convex_hull_features_6features.m
- mergepts.m
- moment_norm.m
- connectspline.m
- kmeans_2class_kNearest_subcluster.m
- computeVectorDistance2class.m

The 'compute_convex_hull_features_36Points.m' was MATLAB script which read the co-ordinates of vertebra points from .shp36 file. This was done over a large number of files stored in the local directory. For every set of such vertebra points, the 'compute_convex_hull_features_6features' function was called from its corresponding M-file 'compute_convex_hull_6features.m'. This function in all calculates six features and the image area of the vertebra and returns to the calling script. The features calculated for all the vertebrae are then stored in an Excel Spread Sheet (xls).
The function defined in mergepts.m and connectspline.m was used in the compute_convex_hull_features_36Points.m and the function defined in moment_norm was used by the compute_convex_hull_6features.m.

The 'kmeans_2class_kNearest_subcluster.m' was a MATLAB script which read the features of the vertebrae from the Excel Spread Sheet. It also read variables from the 'Parameters_traction.mat' file (MAT-file) which stored the trained clusters for classification of vertebrae on the basis of presence or absence of Traction. It also included the mean and standard deviation values for all features. Each set of features (corresponding to one vertebra) are normalized using the mean and standard deviation values from the MAT-file. This normalized set of features along with the cluster centers for classifications are given as input to the function defined in 'computeVectorDistance2class.m'. In this function the actual classification takes place. With the normalized features it checks the distances of each feature to it

For the project KMeans Classifier for detecting traction, the above MATLAB code needed to be converted to C++. It was also required that the classification be done for one given vertebra at a time. The working of the code explained above worked on a large set of vertebrae images in Batch-mode. Hence, the two files that were changed from the above code were:

- compute_convex_hull_features_36Points.m
- kmeans_2class_kNearest_subcluster.m

The changes made to compute_convex_hull_features_36Points.m, so that it worked on a single vertebra were made and stored in the same file. The code of kmeans_2class_kNearest_subcluster.m was rewritten in C++ which worked upon a set of features for only one vertebra. This code then calls the computeVectorDistance2class function.

Also, in the provided workspace calculated the six features were calculated and stored in an Excel Spread Sheet. Then, the Kmeans classifier script was run for classifying by reading back the features from the Excel Spread Sheet. This project calculates the six features and then run the classification program on it, thus eliminating the use the Excel Spread Sheet.

A.3.1. Creating the Dynamic Linked Libraries for using the MATLAB routines:

All the files described above are kept in a single folder. In the MATLAB compiler, set the current working directory to the directory where all the files are stored and then, run the command:

```
mcc  -W  lib:libfcmt  -T  link:lib  compute_convex_hull_features_36Points
computeVectorDistance2class
```

After running this command, the files that are generated in the current directory are:

- libfcmt.c
- libfcmt.h
- libfcmt_mcc_component_data.c
- libfcmt.dll
- libfcmt.lib
- libfcmt.exports
- libfcmt.exp
- libfcmt.ctf

A.3.2. Creating the workspace in C/C++ developer environment with the libraries included:

In Microsoft Visual Studio .NET 2005, create a new Win32 Console project. Include the following files in the source code:

```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
#include"mat.h"
#include"libfcmt.h"
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libfcmt.lib")
```

The C++ implementation was divided into these parts:

1) Reading the MAT-file and extracting the cluster centers for classification and extracting the mean and standard deviation values for these features.

2) Reading the .shp36 files to get the vertebra points and then calling an appropriate MATLAB routine to calculate features.
3) Calling the MATLAB routine to classify for the presence or absence of traction based on the values of these features.

The file name of the .shp36 file was an input parameter taken as a command line argument. The filename to be passed here was to be the absolute path on the local machine or a relative path can be given if the .shp36 file in stored in the same or one the subfolders of the current workspace. Hence, the process can be worked on any .shp36 file just by passing different file names at different calls.

A class called ‘Data’ was created which stores all the values of features for a vertebra also it stores all the values extracted from the MAT file, which are the centers of the classification clusters and the mean and standard deviation values of the features. Several functions are defined that work on these member variables. Keeping a separate class for the features values and for the classification parameters was considered redundant as the program was just suppose to work on one given vertebra at a time.

Since, there are more than one function calls to the MATLAB routines, the initialization and termination of the MATLAB compiler are done close to the entry and exit points of the C++ program.

The call to the program was given on the command prompt as:

```
C:\<WorkSpaceDir>\KMeansClassifier_Traction C:\vertebra\C01235_3.shp36
```

A.4. K-Means Classifier for detecting Anterior Osteophytes

The K-Means Classifier for detecting Osteophytes was a project which calculates features on single vertebra and then applies K-Means clustering technique to classify it as NORMAL or ABNORMAL (Severe or Moderate) for the presence or absence of Osteophytes.

The initial workspace that was provided included the following files:

- compute_convex_hull_features_36Points.m
- compute_convex_hull_features_6features.m
- mergepts.m
- moment_norm.m
- connectspline.m
- kmeans_2class_kNearest_subcluster.m
- computeVectorDistance3class.m

The 'compute_convex_hull_features_36Points.m' was MATLAB script which read the co-ordinates of vertebra points from .shp36 file. This was done over a large number of files stored in the local directory. For every set of such vertebra points, the 'compute_convex_hull_features_6features' function was called from its corresponding M-file 'compute_convex_hull_6features.m'. This function in all calculates six features and the image area of the vertebra and returns to the calling script. The features calculated for all the vertebrae are then stored in an Excel Spread Sheet (xls).
The function defined in mergepts.m and connectspline.m was used in the compute_convex_hull_features_36Points.m and the function defined in moment_norm was used by the compute_convex_hull_6features.m.

The 'kmeans_2class_kNearest_subcluster.m' was a MATLAB script which read the features of the vertebrae from the Excel Spread Sheet. It also read variables from the 'Parameters_Osteophytes.mat' file (MAT-file) which stored the trained clusters for classification of vertebrae on the basis of presence or absence of Osteophytes. It also included the mean and standard deviation values for all features. Each set of features (corresponding to one vertebra) are normalized using the mean and standard deviation values from the MAT-file. This normalized set of features along with the cluster centers for classifications are given as input to the function defined in 'computeVectorDistance3class.m'. In this function the actual classification takes place. With the normalized features it checks the distances of each feature to it

For the project KMeans Classifier for detecting Osteophytes, the above MATLAB code needed to be converted to C++. It was also required that the classification be done for one given vertebra at a time. The working of the code explained above worked on a large set of vertebrae images in Batch-mode. Hence, the two files that were changed from the above code were:

- compute_convex_hull_features_36Points.m
- kmeans_2class_kNearest_subcluster.m

The changes made to compute_convex_hull_features_36Points.m, so that it worked on a single vertebra were made and stored in the same file. The code of kmeans_2class_kNearest_subcluster.m was rewritten in C++ which worked upon a set of features for only one vertebra. This code then calls the computeVectorDistance3class function.

Also, in the provided workspace calculated the six features were calculated and stored in an Excel Spread Sheet. Then, the Kmeans classifier script was run for classifying by reading back the features from the Excel Spread Sheet. This project calculates the six features and then run the classification program on it, thus eliminating the use the Excel Spread Sheet.

A.4.1. Creating the Dynamic Linked Libraries for using the MATLAB routines:

All the files described above are kept in a single folder. In the MATLAB compiler, set the current working directory to the directory where all the files are stored and then, run the command:

```
mcc -W lib:libfcmo -T link:lib compute_convex_hull_features_36Points
computeVectorDistance3class
```

After running this command, the files that are generated in the current directory are:

- libfcmo.c
- libfcmo.h
- libfcmo_mcc_component_data.c
- libfcmo.dll
- libfcmo.lib
- libfcmo.exports
- libfcmo.exp
- libfcmo.ctf

A.4.2. Creating the workspace in C/C++ developer environment with the libraries included:

In Microsoft Visual Studio .NET 2005, create a new Win32 Console project. Include the following files in the source code:

```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
#include"mat.h"
#include"libfcmo.h"
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libfcmo.lib")
```

The C++ implementation was divided into these parts:

1) Reading the MAT-file and extracting the cluster centers for classification and extracting the mean and standard deviation values for these features.

2) Reading the .shp36 files to get the vertebra points and then calling an appropriate MATLAB routine to calculate features.
3) Calling the MATLAB routine to classify for the presence or absence of osteophytes based on the values of these features.

The file name of the .shp36 file was an input parameter taken as a command line argument. The filename to be passed here was to be the absolute path on the local machine or a relative path can be given if the .shp36 file in stored in the same or one the subfolders of the current workspace. Hence, the process can be worked on any .shp36 file just by passing different file names at different calls.

A class called 'Data' was created which stores all the values of features for a vertebra also it stores all the values extracted from the MAT file, which are the centers of the classification clusters and the mean and standard deviation values of the features. Several functions are defined that work on these member variables. Keeping a separate class for the features values and for the classification parameters was considered redundant as the program was just suppose to work on one given vertebra at a time.

Since, there are more than one function calls to the MATLAB routines, the initialization and termination of the MATLAB compiler are done close to the entry and exit points of the C++ program.

The call to the program was given on the command prompt as:

```
C:\<WorkSpaceDir>\KMeansClassifier_Osteophytes C:\vertebra\C01235_3.shp36
```

APPENDIX B

Read- Me file for the project ‘NewDiscSpaceNarrowing’

B.1. New Disc Space Narrowing using Self Organizing Maps

The New Disc Space Narrowing was a project which calculates features on a pair of consecutive vertebrae and then using K-means and a Self Organizing Map clustering technique classifies the degree of disc space narrowing into four grades (0-3), where 0 represents normal spacing and 3 represents significant narrowing.

The initial workspace that was provided included the following files:

- mainPairVertebraBoundaryPoints.m
- generateVertebraBoundaryPair.m
- connect_spline.m
- KMeansModel_individualTest.m
- discSpaceNarrowing a VC++ project workspace.

The 'mainPairVertebraBoundaryPoints.m' was a MATLAB script which reads the co-ordinates of vertebra boundary points of two vertebrae from their respective .shp36 files. The 'generateVertebraBoundaryPair.m' and 'connect_spline.m' routines are called by this function to generate the complete boundary of each vertebra. This function saved the coordinate points of the complete boundary into a text file.

The discSpaceNarrowing VC++ project workspace reads this text file to get the complete boundary of the two vertebrae under analysis. It then computes the four Disc Space Narrowing features for the given pair of vertebrae. The computed features are written to another text file.

The 'KMeansModel_individualTest.m' was a MATLAB script which was used to classify the given DSN features of a pair of vertebrae according to the degree of disc space narrowing. It grades the features between 0-3, where 0 signifies a normal spacing between the pair of vertebrae and 3 represents substantial narrowing.

The inputs to the 'KMeansModel_individualTest.m' script are the DSN features which are read from the output file generated by the VC++ workspace 'discSpaceNarrowing'. This routine also requires an already trained model to test the new set of features for the purpose of classifying them. The trained model was stored in a MAT-file stored in the same directory. Two other MAT-files provide the mean and the standard deviation values of all the DSN features which are used to normalize the input feature vector.

In the project New Disc Space Narrowing, the given workspace functions were linked so that at every run of the solution, the disc space narrowing features were calculated for a given pair of vertebra, its classification according to the Kmeans and Self organizing maps was done and finally it generated a grade (0-3) as its output, specifying the degree of disc space narrowing.

The following M-files were changed:

- mainPairVertebraBoundaryPoints.m
- KMeansModel_individualTest.m

Both these Matlab scripts were changed to Matlab functions by putting the code within the scripts into a wrapper function. Now these functions are called from the VC++ workspace by creating a DLL for them. Creation of the DLLs and the new VC++ workspace are explained below:

B.1.1. Creating the Dynamic Linked Libraries for using the MATLAB routines:

All the files described above are kept in a single folder. In the MATLAB compiler, set the current working directory to the directory where all the M-files are stored and then, run the command:

```
mcc -W lib:libdsn -T link:lib kmeansmodeltest
                                      generatevertebraboundarypairshape36
```

After running this command, the files that are generated in the current directory are:

- libdsn.c
- libdsn.h
- libdsn_mcc_component_data.c
- libdsn.dll
- libdsn.lib
- libdsn.exports
- libdsn.exp
- libdsn.ctf

B.1.2. Creating the workspace in C/C++ developer environment with the libraries included:

In Microsoft Visual Studio .NET 2005, create a new Win32 Console project; copy the discSpaceNarrowing VC++ workspace files into this new project. Include the following files in the source code files.

```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
#include"mat.h"
#include"libdsn.h"
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libdsn.lib")
```

The header files and libraries ‘libmat’ and ‘libmx’ are found in the <matlab_root>\extern\include and <matlab_root>\extern\lib. The files generated by the MATLAB compiler on the ‘mcc’ command are copied to the current directory of this VC++ workspace.

The code of the existing discSpaceNarrowing VC++ workspace used dynamic memory allocations, due to which warnings may be generated while running the same code in the NewDiscSpaceNarrowing VC++ project. To disable these warnings, open the 'NewDiscSpaceNarrowing' project in Visual Studio .NET 2005 (or any other environment), in the 'Solution Explorer' frame, right click on 'NewDiscSpaceNarrowing', now click 'Properties'. In the window that opens up, go to ConfigurationProperties>C/C++>CodeGeneration. Now click on tab (on the right side) 'Basic Runtime Checks' and change its value to 'Stack Frames (/RTCs)' by selecting it from the drop-down menu. Click Apply and OK.

The main() function in the discSpaceNarrowing VC++ workpace was changed. It was converted to a function: DiscSpaceNarrowing_c(Pair*) and another file was created named 'NewDiscSpaceNarrowing.cpp' where the entry point i.e. main() function of the program was placed from where the function call to DiscSpaceNarrowing_c() was made.

A class named 'Pair' was created which encapsulates all features and necessary data related to a pair of vertebra, which are needed for the calculation of the DSN features and it's grading. All newly added functions are also encapsulated in this class.

The file names of the .shp36 files of the vertebrae under investigation are input parameters taken as command line arguments. These command line arguments are the absolute paths of these .shp36 files on the local machine or relative paths can be given if the .shp36 files are stored in the same or one the subfolders of the current workspace.

Since, there are more than one function calls to several MATLAB routines, the initialization and termination of the MATLAB compiler are done close to the entry and exit points of the C++ program.

The call to the program was given on the command prompt as:

```
C:\<WorkSpaceDir>\NewDiscSpaceNarrowing C:\vertebra\C01235_3.shp36
                                        C:\vertebra\C01235_4.shp36
```

APPENDIX C

Read- Me file for the project ‘NewSubluxation’

C.1. NewSubluxation using a neural network

The NewSubluxation was a project which calculates features on a group of adjacent cervical vertebrae and then simulates a neural network to calculate a score for the given group of the cervical vertebrae.

The initial workspace that was provided included the following files:

- subluxationFeatures.m
- getScore.m
- connectspline.m

The 'subluxationFeatures.m' was a MATLAB script which did the major computational part of the feature calculations. It read the images of adjacent vertebrae from the local machine, and it then generated an image consisting of all the cervical vertebrae by logically ORing each of the input images. Several image processing tools were applied to the image of the complete cervical vertebrae. Also, the centroids and areas of each vertebra that make up the complete image were calculated and used for feature calculation.

The features calculated were output to a text file on the local machine. Also, the 'subluxationFeatures.m' script worked on several groups of cervical vertebrae in batch mode. The getScore.m was a MATLAB script that runs the simulation of a neural network using the inbuilt 'sim' function defined in the Neural Network toolbox. The MATLAB function connectspline.m was called by the subluxationFeatures.m to generate a more complete boundary of a vertebra based on the inputs of a .shp36 file.

In the NewSubluxation project, the given M-files needed to be linked so that on every run of the solution, the subluxation features are calculated for a given group of adjacent cervical vertebrae, and a score was generated for it.

The following M-files were changed:

- subluxationFeatures.m
- getScore.m

Both these Matlab scripts were changed to Matlab functions by putting the code within the scripts into a wrapper function. An additional MATLAB function 'compute_subluxationFeatures.m' was created, this was the function which was called from VC++, it processes input arguments received and then calls the function in 'subluxationFeatures.m'. Creation of the DLLs and the new VC++ workspace are explained below:

C.1.1. Creating the Dynamic Linked Libraries for using the MATLAB routines:

All the files described above are kept in a single folder. In the MATLAB compiler, set the current working directory to the directory where all the M-files are stored and then, run the command:

```
mcc -W lib:libsublx -T link:lib compute_subluxationFeatures getScore
```

After running this command, the files that are generated in the current directory are:

- libsublx.c
- libsublx.h
- libsublx_mcc_component_data.c
- libsublx.dll
- libsublx.lib
- libsublx.exports
- libsublx.exp
- libsublx.ctf

The 'getScore.m' function uses the 'sim.m' function defined in the Neural Network toolbox of MATLAB. By generating the libraries (libsulx files), a C/C++ interface was created that can be used to call the functions compute_subluxation.m and getScore.m. These functions can use all the MATLAB built-in functions within their codes. Although, the use of functions from the Neural Network toolbox can generate warnings as these are not included in the MATLAB compiler which gets loaded via the uses of these libraries. Hence, to alleviate this problem, we add (copy) all the functions of the Neural Network toolbox in the same directory where the other M-functions are stored.

These functions are found in the local directory (on a machine where MATLAB was installed) : <matlabroot>\nnet\nnet\@network\

The files that are to be copied are:

- adapt.m
- disp.m
- display.m
- gensim.m
- init.m
- loadobj.m
- network.m
- revert.m
- sim.m
- train.m

Two additional files can be found in the same directory which are: 'subasgn.m' and 'subsref.m'. These files are necessarily not to be copied to our current directory with other M-files, this compulsion was put because the functions defined in these two M-files are not required in our implementation and can generate warnings while creating the libraries and using them in our C++ program.

Now with all the required M-functions placed in one folder, generate the required 'libsulx' libraries with the –mcc command provided above.

C.1.2. Creating the workspace in C/C++ developer environment with the libraries included:

In Microsoft Visual Studio .NET 2005, create a new Win32 Console project NewSubluxation. Include the following files in the source code files:

```
#include<iostream>
#include<cmath>
#include<string>
#include<cstring>
#include"mat.h"
#include"libsublx.h"
#pragma comment(lib, "libmat.lib")
#pragma comment(lib, "libmx.lib")
#pragma comment(lib, "libsublx.lib")
```

The header files and libraries named 'libmat' and 'libmx' are found in the <matlab_root>\extern\include and <matlab_root>\extern\lib. The files generated by the MATLAB compiler on the 'mcc' command are copied to the current directory of this VC++ workspace.

The #pragma directives declared above can be avoided in the source code, if these library files are included that was added to the Solution in VC++.

The NewSubluxation project works on several adjacent cervical vertebrae to calculate the subluxation features and to calculate a score based on these features. Hence, the execution of NewSubluxation project requires the filenames of the vertebrae files (.shp36 files). The file names of these .shp36 files of the vertebrae under investigation are provided as input parameters taken as command line arguments. Each of these command line arguments are the absolute paths of these .shp36 files on the local machine or relative paths can be given if the .shp36 files are stored in the same or one of the subfolders of the current workspace.

The main() function that was the entry point of the NewSubluxation project's code encapsulates these filenames into list and also initializes a string containing the filename of the trained model to be used while calculating the score for the features that will be calculated. Hence, a different trained model can be used by changing this value in the code. It then calls the NewSubluxation() function with the list of vertebrae filenames and the trained model's filename as input. The number of input arguments can vary, for the purpose of feature calculation, it was required that either four (C3-C6) or five (C3-C7) filenames of adjacent vertebrae be passed. The trained model included here, will test the features generated for vertebrae C3-C6 irrespective of the number of filenames passed for feature calculation.[1]

A class named 'Cervicals' and a class named 'Vertebra' are created for the implementation of this project. The Vertebra class encapsulates all the properties of a single vertebra like boundary points, etc and necessary functions to operate on them. The Cervicals class encapsulates a list of objects of the Vertebra class and other data related to this group of vertebrae required to calculate the subluxation features and the score. All newly added functions are also encapsulated in these classes.

Since, there are more than one function calls to several MATLAB routines, the initialization and termination of the MATLAB compiler are done close to the entry and exit points of the C++ program.

When required to calculate the subluxation features on vertebra C3-C6, the call to the program was given on the command prompt was as:

```
> C:\<WorkSpaceDir>\NewSubluxation C:\vertebra\C01235_3.shp36 C:\vertebra\C01235_4.shp36
                        C:\vertebra\C01235_5.shp36 C:\vertebra\C01235_6.shp36
```

[1] The trained model can be replaced, by changing the filename that was initialized in the main() routine.

BIBLIOGRAPHY

[1] Fact Sheet: *Osteoarthritis*. American College of Rheumatology, Atlanta, GA, 1994.

[2] Cherkuri M., Stanley R.J., Long L.R., Antani S.K., Thoma G.R. "*Anterior osteophyte discrimination in lumbar vertebrae using size-invariant features.*" Computerized Medical Imaging and Graphics 2004; 28(1/2), pp. 99–108

[3] Stanley R.J., Antani S.K., Long L.R., Thoma G.R., Gupta K., Das M. *"Size-invariant descriptors for detecting regions of abnormal growth in cervical vertebrae."* Computerized Medical Imaging and Graphics; 32(1) (2007), pp. 44–52.

[4] Long L.R. and Thoma G.R., *"Image query and indexing for digital x-rays."* In Proc. SPIE Conference on Storage and Retrieval for Image and Video Databases VII, San Jose, CA, 1999; vol. 3656, pp. 12–21.

[5] Heggeness M.H., Doherty B.J. *"Morphologic study of lumbar vertebral osteophytes."* South Med J 1998; 91(2), pp. 187–9.

[6] Pate D., Goobar J., Resnick D., Haghighi P., Sartoris D.J., Pathria M.N. *"Traction osteophytes of the lumbar spine: radiographic-pathologic correlation.*" Radiology 1988; 166(3), pp. 843–6.

[7] Stanley R.J. and Long L.R. *"A radius of curvature-based approach to cervical spine vertebra image analysis."* Copper Mountain, CO. Proc 38th Annual Rocky Mountain Bioengineering Symposium 2001; 37, pp. 385–390.

[8] Tsai M.D., Jou S.B., Hsieh M.S. *"A new method for lumbar herniated inter-vertebral disc diagnosis based on image analysis of traverse sections.*" Computerized Medical Imaging and Graphics 2002; 26(6), pp. 369–380.

[9] Manber U, *Introduction to Algorithms: A Creative Approach*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA; 1989.

[10] Barber C., Dobkin D., Huhdanpaa H. *"The Quickhull algorithm for convex hulls."* ACM Transactions on Mathematical Software 1996; 22(4), pp. 469–83.

[11] Horn B. *Robot Vision*. 1st ed. MIT Press; Cambridge, MA; 1986. ISBN: 0-262-08159-8.

[12] Chiu S. *"Fuzzy model identification based on cluster estimation.*" Intell Fuzzy Syst J 1994; 2(3), pp. 267–78.

[13] Yager R. and Filev D. *"Generation of fuzzy rules by mountain clustering."* Intell Fuzzy Syst J 1994; 2(3), pp. 209–19.

[14] Seber G.A.F. *Multivariate observations*. Wiley; New York; 1984.

[15] Han J. and Kamber M. *Data mining: Concepts and Techniques*. Morgan-Kaufman; San Francisco, CA; 2000. ISBN: 1-55860-489-8.

[16] The Matlab Online Reference, MathWorks Inc. Product Doucumentation, R2007b. "*http://www.mathworks.com/access/helpdesk/help/techdoc/index.html".*

[17] Gonzalez R. and Woods R. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Boston, MA., 1992. ISBN: 0201508036.

[18] Hanley J.A. and McNeil B.J., *"The meaning and use of the area under a Receiver Operating Characteristic (ROC) curve."* Radiology, 1982; 143(1), pp. 29–36.

VITA

The author, Mohammed Sadiq Das, was born on March 27, 1985 in the city of Bombay, India where he grew up and received his primary and secondary education from Bhavans A. H. Wadia High School. With an interest in the subjects of Mathematics and Computers, a desire to experiment with gadgets and devices, and an inclination towards computer programs he furthered his studies in fields of science and engineering at the University of Mumbai (Bombay), India to receive his Bachelor of Engineering in Information Technology in June 2006. Here, he had opportunities to learn and showcase his knowledge in areas of Component based Software Development, Networking and Image Processing. With a desire to learn more at core of the technologies, he then went on to pursue an M.S. degree in Computer Science at the University of Missouri-Rolla. During the course of this graduate studies, he had the opportunity to research intensively on various aspects of Medical Image Processing and Pattern classification under the supervision of Dr. Ronald Joe Stanley. In his free time, he enjoys playing soccer and chess, and has also had the opportunity to represent his college soccer team at various levels.