ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΓΕΝΙΚΟ ΤΜΗΜΑ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΕΦΑΡΜΟΣΜΕΝΕΣ ΕΠΙΣΤΗΜΕΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΔΙΑΤΡΙΒΗ ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ
ΚΑΤΕΥΘΥΝΣΗ : «ΕΦΑΡΜΟΣΜΕΝΑ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΑ ΜΑΘΗΜΑΤΙΚΑ»

ΜΙΑ ΔΙΕΥΡΥΜΕΝΗ ΜΕΘΟΔΟΣ ΤΑΥΤΙΣΗΣ ΕΙΚΟΝΩΝ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΣΠΑΝΑΚΗΣ

Επιβλέπων : Επίκ. Καθηγητής **Εμ. Μαθιουδάκης**

ΧΑΝΙΑ , 2014

Η διατριβή αυτή εξετάστηκε επιτυχώς από την παρακάτω Τριμελή Επιτροπή

- Επικ. Καθηγητή Εμμανουήλ Μαθιουδάκη
- Δρ. Κ. Μαριάς Ερευν. Β Ινστιτούτο Πληροφορικής-ΙΤΕ
- Δρ. Γ. Νότας Λέκτορας Ιατρικής Σχολής Πανεπιστημίου Κρήτης

η οποία ορίστηκε κατά τη 366/9-10-2013 συνεδρίαση της Συγκλήτου Πολυτεχνείου Κρήτης.

Contents

List of Figures

Περίληψη

Σε αυτή τη διατριβή παρουσιάζεται η εφαρμογή μίας μεθόδου ταύτισης εικόνων βασισμένη στην μεγιστοποίηση της αμοιβαίας πληροφορίας. Πιο συγκεκριμένα θα περιγραφεί η μέθοδος του Maes για ταύτιση εικόνων και θα επισημανθούν τα προβλήματά υλοποίησής της. Στην συνέχεια θα παρουσιαστεί μια επέκταση αυτής της μεθόδου.

Αναλυτικότερα τώρα, ταύτιση ονομάζεται η διαδικασία μετασχηματισμού ενός συνόλου δεδομένων σε ένα άλλο σύνολο. Τα δεδομένα αυτά μπορεί να είναι φωτογραφίες, δεδομένα προερχόμενα από διαφορετικές πηγές, χρονικές στιγμές ή/και συνθήκες φωτισμού. Η ταύτιση εικόνων εφαρμόζεται σε αρκετούς τομείς όπως υπολογιστική όραση, ιατρικές απεικονίσεις και ανάλυση εικόνων από δορυφόρους. Υπάρχουν πολλές μέθοδοι για την ταύτιση εικόνων και οι βασικές τους κατηγορίες είναι οι εξής:

- Feature-based
- Intensity-based

Στην πρώτη κατηγορία η ταύτιση γίνεται βάσει χαρακτηριστικών των εικόνων, τα οποία μπορεί να είναι συγκεκριμένα σημεία ή γραμμές βάσει των οποίων βρίσκουν την αντίστοιχία μεταξύ των συνόλων των χαρακτηριστικών των δύο εικόνων. Οι μέθοδοι της δεύτερης κατηγορίας, στην οποία ανήκει η μέθοδος του Maes, ταυτίζουν ολόκληρες εικόνες ή υποεικόνες συγκρίνοντας τα ¨μοτίβα' των εντάσεων μέσω μετρικών συσχέτισης. Σε αυτές τις μεθόδους ο μετασχηματιμός ο οποίος μετατρέπει την μεταβαλλόμενη εικόνα στην εικόνα αναφοράς είναι αυτός για τον οποίο ο συσχετισμός μεταξύ των δύο εικόνων μεγιστοποιείται. Οι μέθοδοι της πρώτης κατηγορίας είναι γρήγορες, αλλά τείνουν να είναι περισσότερο εξειδικευμένες, ενώ οι πρώτες είναι γενικευμένες περισσότερο ακριβείς, αλλά λόγω του ότι εξετάζουν τις εικόνες pixel προς pixel, τείνουν να είναι πιο αργές. Αυτός είναι ο λόγος για τον οποίο γίνεται η μελέτη της ταύτισης των εικόνων, η οποία παραμένει κατ΄ ουσίαν ένα άλυτο πρόβλημα.

Η μέθοδος του Maes χρησιμοποιεί ως μετρική συσχέτισης την αμοιβαία πληροφορία. Για την κατανόηση της αμοιβαίας πληροφορίας, χρειάζεται να κατανοηθεί ο όρος εντροπία. Εντροπία είναι ένα στατιστικό μέγεθος, όπως η μέση τιμή και η διακύμανση, η οποία δείχνει ποσο ¨τυχαία' είναι μια τυχαία μεταβλητή. Μεγάλη εντροπία υποδηλώνει μεγάλη τυχαιότητα, ενώ αντιστρόγως μικρή εντροπία δείχνει μικρή τυχαιότητα. Η εντροπία μιας τυχαίας μεταβλητής ορίζεται ως εξής:

$$H(X) = -\sum_{x \in X} p_X(x) \log_2 p_X(x) \tag{1}$$

Ομοίως ορίζεται και εντροπία για περισσότερες από μία μεταβλητές. Για δύο μεταβλητές, η κοινή και η υπο συνθήκη εντροπία ορίζονται αντίστοιχα ως εξής:

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in y} p(x,y) \log_2 p(x,y) \tag{2}$$

$$H(Y|X) = -\sum_{x\in X}\sum_{y\in y} p(x,y)\log_2 p(y|x) \tag{3}$$

Η πρώτη δείχνει την τυχαιότητα των δύο μεταβλητών μαζί, ενώ η δεύτερη δείχνει πόσο τυχαία είναι η μεταβλητή Υ, δεδομένης της γνώσης της Χ. Η αμοιβαία πληροφορία ορίζεται ως εξής:

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \tag{4}$$

Ο Maes θεωρεί τις δύο εικονες, με όνομα **F** την μεταβαλλόμενη εικόνα και **R** την εικόνα αναφοράς, ως δύο τυχαίες μεταβλητές και τις τιμές των εικόνων στα διάφορα pixels τις τιμές των μεταβλητών αυτών. Για τον υπολογισμό της αμοιβαίας πληροφορίας, χρειάζεται η γνώση των κατανομών των ¨τυχαίων' μεταβλητών, η οποία αποκτάται με την κατασκευή ενός κοινού ιστογράμματος $h(f(s), r(Ts)$, όπου $f(s)$ η τιμή της μεταβαλλόμενης εικόνας στο pixel s και $r(Ts)$ η τιμή της εικόνας αναφοράς στο pixel Ts. Ο μετασχηματισμός των συντεταγμένων $P_{\mathbf{F}}$ του pixel s στις συντεγμένες $P_{\mathbf{R}}$ του pixel Ts είναι ο ακόλουθος:

$$V_{\mathbf{R}} \cdot (P_{\mathbf{R}} - C_{\mathbf{R}}) = V_{\mathbf{F}} \cdot R \cdot (P_{\mathbf{F}} - C_{\mathbf{F}}) + t \tag{5}$$

όπου $C_{\mathbf{R}}, C_{\mathbf{F}}$ οι συντεταγμένες των κέντρων των εικόνων R, F αντίστοιχα, $V_{\mathbf{R}}, V_{\mathbf{F}}$ οι 2×2 πίνακες μεγέθους των pixel των εικόνων R, F αντίστοιχα, R ο 2×2 πίνακας περιστροφής και $t = (t_x, t_y)$ το διάνυσμα μετατόπισης. Οι συντετεγμένες του Ts μπορεί να μήν είναι ακέραιοι αριθμοί. Για τον υπολογισμό του $r(Ts)$ μπορούμε να την προσεγγίσουμε, είτε βάσει του πλησιέστερου γείτονα, το οποίο όμως μπορεί να προκαλέσει απότομες μεταβολές στον υπολογισμό της αμοιβαίας πληροφορίας όταν μετακινείται η εικόνα, είτε μέσω διγραμμικής παρεμβολής, η οποία όμως εισάγει νέες εντάσεις στο ιστόγραμμα οι οποίες κανονικά δεν υπάρχουν. Υπάρχουν τρεις τρόποι για την κατασκευή του κοινού ιστρογράμματος.

- Πλησιέστερου Γείτονα: Η τιμή $r(Ts)$ είναι ίση με την τιμή του πλησιέστερουpixel. Αυτή η στρατηγική είναι η απλούστατη και η λιγότερο ακριβής σε επίπεδο subpixel.
- Τριγραμμική (σε 2D διγραμμική) Παρεμβολή: Εισάγει νέες τιμές στην εικόνα αναφοράς, οδηγώντας σε απρόβλεπτες αλλαγές στην κατανομή $p_R(r)$.
- Τριγραμμική (σε 2D διγραμμική) Partial Volume Distribution: Η συμβολή της τιμής $f(s)$ στο κοινό ιστόγραμμα διαμοιράζεται στους οκτώ (τέσσερις στις 2D) πλησιέστερους γείτονες.

Οι κατανομές υπολογίζονται ως εξής:

$$p_{\mathbf{F},\mathbf{R}}(f,r) = \frac{h(f,r)}{\sum_{i\in\mathbf{F}, j\in\mathbf{R}} h(i,j)} \tag{6}$$

$$p_{\mathbf{F}}(f) = \sum_{r\in\mathbf{R}} p_{\mathbf{F},\mathbf{R}}(f,r) \tag{7}$$

$$p_{\mathbf{R}}(r) = \sum_{f \in \mathbf{F}} p_{\mathbf{F},\mathbf{R}}(f, r) \tag{8}$$

Βάσει των παραπάνω κατανομών, υπολογίζεται η αμοιβαία πληροφορία των εικόνων. Για την εύρεση του βέλτιστου μετασχηματισμού, χρησιμοποιείται η μέθοδος του Powell η οποία βρίσκει ακρότατα μίας συνάρτησης, χρησιμοποιώντας την μέθοδο του Brent. Ο αλγόριθμος του Powell αν και βρίσκει ακρότατα, πολύ πιθανόν αυτά να είναι τοπικά. Παίζει μεγάλο βαθμό η μορφή της συνάρτησης. Αν η συνάρτηση έχει λίγα τοπικά ακρότατα ή/και είναι αρκετά ομαλή, τότε είναι δυνατόν να βρεθεί σχετικά εύκολα χωρίς να χρειαστεί να ψάξουμε το κατάλληλο αρχικό σημείο και τα κατάλληλα αρχικά διανύσματα κατεύθυνσης. Στην αντίθετη περίπτωση, θα πρέπει να γίνει πολύ προσεκτική επιλογή του αρχικού σημείου και των αρχικών διανυσμάτων κατεύθυνσης προκειμένου να βρούμε το ολικό ακρότατο. Το πρόβλημα αυτό εμφανίζεται στην μέθοδο του Maes. Η μορφή της συνάρτησής μας (δηλαδή η αμοιβαία πληροφορία) εξαρτάται από τις εικόνες προς ταύτιση. Στην περίπτωση που οι εικόνες είναι αρκετά καλής ποιότητας και δεν εμφανίζουν συμμετρίες στο χώρο τους, τότε η αμοιβαία πληροφορία (υπο την μορφή συνάρτησης με όρισμα τον μετασχηματισμό T) είναι αρκετά ομαλή με σχετικά λίγα ακρότατα οπότε δύναται να βρεθεί εύκολα και γρήγορα ο βέλτιστος μετασχηματισμός. Στην περίπτωση όμως που η εικόνα δεν έχει καλή ανάλυση με πολλές συμμετρίες, τότε η αμοιβαία πληροφορία παρουσιάζει πολλά ακρότατα οπότε χρειάζεται πολύ καλή επιλογή του αρχικού μετασχηματισμού και των αρχικών διανυσμάτων κατεύθυνσης για να βρεθεί ο (ολικά) βέλτιστος μετασχηματιμός. Μάλιστα σε ορισμένα πειράματα, ο ¨βέλτιστος' μετασχηματιμός μετέθετε την εικόνα περισσότερο από όσο είναι δυνατόν (για παραδείγμα η μεταβαλλόμενη εικόνα διάστασης 128×128, χρειαζόταν να μετακινηθεί κατά 140 pixels για να ταιριάξει με την εικόνα αναφοράς). Στην βελτιωμένη μέθοδο συνεχίζεται να χρησιμοποιείται η αμοιβαία πληροφορία, απλά αλλάζει ο τρόπος αναζήτησης.

Η ιδέα είναι να χρησιμοποιηθεί μία μέθοδος, η οποία εγγυάται την εύρεση του βέλτιστου μετασχηματισμού, χωρίς την πιθανότητα να γίνει αναζήτηση σε χώρο όπως στο παραπάνω παράδειγμα. Ο αλγόριθμος που εφαρμόσθηκε για την εύρεση του (ολικά) βέλτιστου μετασχηματισμού είναι ένας γενετικός αλγόριθμος. Οι γενετικοί αλγόριθμοι είναι μια κατηγορία μεθόδων αναζήτησης λύσεων σε ένα χώρο οι οποίοι μιμούνται την διαδικασία της φυσικής επιλογής όπως την διατύπωσε ο Κάρολος Δαρβίνος. Όπως στην φύση τα άτομα του πληθυσμού ενός είδους διασταυρώνονται μεταξύ τους και οι απόγονοι έχουν τα χαρακτηριστικά των γονέων τους και επιβιώνουν τα άτομα με τα καλύτερα χαρακτηριστικά, έτσι και εδώ διασταυρώνονται κάποιες από τις υποψήφιες (κατά προτίμηση οι καλύτερες, χωρίς να αποκλείουμε εντελώς τις λιγότερο καλές) λύσεις ενός πληθυσμού και επιβιώνουν αυτές με τα καλύτερα χαρακτηριστικά, ήτοι αυτές για τις οποίες η αμοιβαία πληροφορία μεγιστοποιείται. Έτσι και εδώ χρησιμοποιείται ένας γενετικός αλγόριθμος για να βρούμε έναν πληθυσμό ο οποίος θα περιέχει μια λύση η οποία θα είναι η βέλτιστη. Αναλυτικότερα ένας γενετικός αλγόριθμος έχει τα εξής βήματα:

```
function G.A($obj\_fun$,$m\_rate$,$c\_rate$)
    $Old\_Population \leftarrow Initialize()$
    while true do
        $Selected\_Members = Selection(Old\_Population)$
        $New\_Population \leftarrow Crossbreed(Selected\_Members, c\_rate)$
        $New\_Population \leftarrow Mutate(New\_Population, m\_rate)$
        if criteria met then
            break
        end if
        $Old\_Population \leftarrow New\_Population$
    end while
    return best from New_Population
end function
```

Τα κριτήρια τερματισμού είναι διάφορα:

- Η εύρεση "βέλτιστης" λύσης η οποία ικανοποιεί κάποια κριτήρια μεγίστου/ελαχίστου
- Πεπερασμένος αριθμός γεννεών
- Χειροκίνητη επίβλεψη
- Συνδυασμός των παραπάνω

Στην εργασία αυτή χρησιμοποιήθηκε ως κριτήριο τερματισμού ο πεπερασμένος αριθμός γεννεών. Γενικά, οι γενετικοί αλγόριθμοι, μετά από πολλές γεννεές και ειδικά αν είναι ο χώρος αναζήτησης της λύσης σχετικά περιορισμένος, τότε είναι δυνατόν να βρεθεί ο (ολικά) βέλτιστος μετασχηματισμός. Το βασικό πρόβλημα είναι ότι είναι χρονοβόρος ειδικά σε περισσότερες διαστάσεις και όταν η συνάρτηση που θέλουμε να μεγιστοποιήσουμε είναι υπολογιστικά ακριβή. Επίσης σημαντικό ρόλο για την επιτυχία του γενετικού αλγορίθμου παίζουν ο ρυθμός ματάλλαξης και η πιθανότητα διασταύρωσης. Όσο μεγαλύτερη πιθανότητα διασταύρωσης, τόσο πιο εύκολα νέοι απόγονοι παράγονται, ενώ ο ρυθμός μετάλλαξης διαφοροποεί περαιτέρω τον πληθυσμό αποτρέποντας έτσι την περίπτωση να κολλήσει ο αλγόριθμος σε τοπικό ακρότατο και να παραμείνει εκεί. Ειδικά στην αμοιβαία πληροφορία όπου είναι πολύ πιθανόν να υπάρχει ένας μεγάλος αριθμός τοπικών ακροτάτων, χρησιμοποιήθηκε μεγάλος ρυθμός μετάλλαξης. Τέλος, χρησιμοποιήθηκε η μέθοδος elitism, η οποία μετά τη δημιουργία μιας νέας γενιάς υποψήφιων λύσεων, συγκρίνει την βέλτιστη λύση της προηγούμενης γενιάς με αυτή της νέας γενιάς και αν η πρώτη είναι καλύτερη από τη δεύτερη, τότε εισέρχεται στην νέα γενιά αναλλοίωτη, αντικαθιστώντας μια υποψήφια λύση της νέας γεννιάς (συνήθως την χείριστη). Τα πειράματα της ταύτισης εικόνων υπήρξαν επιτυχή, αλλά χρειάστηκε μεγάλος αριθμός επαναλήψεων για να γίνει αυτό. Έτσι, η μέθοδος αποδεικνύεται χρήσιμη, όταν υπάρχει ανάγκη για υψηλής ποιότητας αποτελέσματα, χωρίς να υπάρχει περιορισμός στο ζήτημα χρόνου/κόστους εκτέλεσης.

Abstract

Image Registration is the process of transforming sets of data acquired at different time-points, sensors and viewpoints into a single coordinate system. It is widely used in computer vision, medical imaging and satellite image analysis. Although it has been a central research topic in computer vision and medical image analysis for a long time, there are still unresolved issues and success rates seem to be data-dependent. There are many categories of methods that are able to align images, but usually they are either specialized and accurate for specific types of data or more generic but slower with a possibility of stumbling upon pitfalls. In this work, we describe our implementation and results on Maes' method By using three different variants of mutual information (used as the similarity measure), we present indicative results from different imaging domains and discuss the drawbacks/pitfalls of the method especially with regard to initial transformation selection and the initial direction vectors. The results are quite accurate with translation and rotation when dealing with images of good quality. However, the choice of a starting point and the initial direction vectors proved to be two critical factors for the success of the method, since different starting point or/and different initial direction vectors may lead to different "optimal" alignment registration results between the images. In order to solve this problem, we propose an extension of this method by enhancing its global optimization scheme by means of stochastic optimization.

Key words: Image Registration, Mutual Information, Genetic Algorithms.

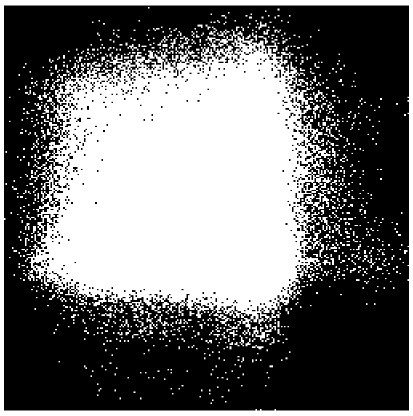
Introduction

The alignment or registration of multi-modal images is an important procedure in many fields e.g medical imaging. For example, when it comes to medical imaging, the images of an object that are acquired from different modes show different things. Therefore, in order to have a good knowledge of the patient's condition, we need to have the images' alignment which is achieved by many registration methods. The major categories of image registration methods are two: the the feature-based methods and intensity-based methods. The first category includes methods that use the features (points, lines or contours) of the images to find the correspondence between the feature sets of them. The correspondence is the geometric transformation of the selected features of the first image to those of the second. The major advantage of the feature-based methods is that they are fast, since they do not use the whole images to in order to align them. Their disadvantage is the difficulty to find proper features for extraction especially when it comes to bad resolution or multi-modal images. On the other hand, the intensity-based methods use correlation metrics to compare image intensity patterns. Their disadvantage is the need to use the whole image or at least a sub image, which makes them slow. Unlike the feature-based, they tend to perform better when the images are of poor quality or multi-modal.
In this paper, an optimized method based on Maes' registration method is presented along with the results. Specifically, in the next section the next section the Maes' method of registration is presented and explained. After the explanation of the method, the experiments are studied in the third section and discuss the disadvantages of the method. In the fourth section, a new method is proposed for image registration and is compared with Maes' one. Finally, the fifth section is the section with conclusions and new approaches are discussed.

Maes' Method

Introduction

Maes' method [1] is an intensity-based method. The idea is that two images that are perfectly aligned have maximum mutual information, while the images that cannot be aligned at all, have none. Finally images that are partially aligned have some mutual information [4]. The goal is to find the transformation that gives the maximum mutual information.


(a) Join Histogram before Registration

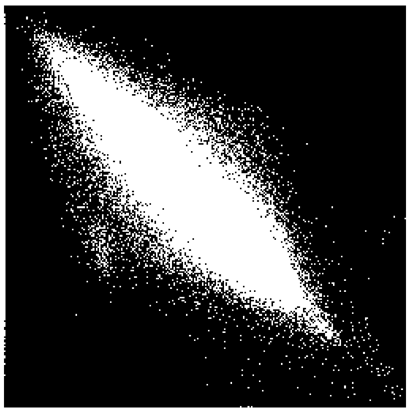

(b) Join Histogram after Registration

Figure 1: Joint Histograms

Mutual information

The first question is: What is mutual information. In order to become more familiar with the concept of mutual information, first, we must understand what entropy is. Entropy is a statistic measure that shows the randomness of a random variable, i.e how chaotic is a random variable. Given a random variable X, its entropy is:

$$H(X) = -\sum_{x \in X} p_X(x) \log_2 p_X(x) \tag{9}$$

A high entropy indicates grater randomness. When there are more than one random variables, we define joint entropy and conditional entropy respectively:

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in y} p(x,y) \log_2 p(x,y) \tag{10}$$

$$H(Y|X) = -\sum_{x \in X} \sum_{y \in y} p(x,y) \log_2 p(y|x) \tag{11}$$

Mutual Information is calculated using the following formula:

$$I(X,Y) = H(X) + H(Y) - H(X,Y)$$

The two images to be registered **F** (Floating image) and **R** (Reference image) are two random variables whose values are the pixel intensities. In order to calculate the images' mutual information, a joint histogram h(f,r) is constructed, where f(s) the intensity of **F** at pixel with coordinates s and r(Ts) the intensity of **R** at pixel with coordinates Ts
where T the transformation operator. The transformation of pixel coordinates $P_{\mathbf{F}}$ of floating image **F** to pixel coordinates $P_{\mathbf{R}}$ of reference image **R** is:

$$V_{\mathbf{R}} \cdot (P_{\mathbf{R}} - C_{\mathbf{R}}) = V_{\mathbf{F}} \cdot R \cdot (P_{\mathbf{F}} - C_{\mathbf{F}}) + t \tag{12}$$

where,$V_{\mathbf{F}}$, $V_{\mathbf{R}}$ the 2×2 matrix that represents the pixel size of the images, R the 2×2 rotation matrix, $t = (t_x, t_y)$ the translation vector and $C_{\mathbf{R}}$, $C_{\mathbf{R}}$ the centers of the images. The problem is that the coordinates of Ts are not integers and therefore Ts lies in the subpixel space. For the solution of this problem, Maes proposed three interpolation strategies, as shown below:

- Nearest Neighbour: The value r(Ts) is equal to the value of the pixel nearest to it. This strategy doesn't provide good subpixel accuracy
- Trilinear (in 2D bilinear) Interpolation: It introduces new values in reference image, leading to unpredictable changes in the distribution $p_R(r)$.
- Trilinear (in 2D bilinear) Partial Volume Distribution: The contribution of f(s) to the joint histogram is distributed to the eight (in 2D four) nearest neighbours.

The construction of the joint histogram we can calculate the marginal and the joint entropy as we see in the following equations:

$$p_{F,R}(f, r) = \frac{h(f, r)}{\sum_{i \in F, j \in R} h(i, j)} \tag{13}$$

$$p_F(f) = \sum_{r \in R} p_{F,R}(f, r) \tag{14}$$

$$p_R(r) = \sum_{f \in F} p_{F,R}(f, r) \tag{15}$$

With these equations, the calculation of the marginal and joint entropy and therefore mutual information can be done. The optimal transformation is found using the multidimensional method of Powell for search of minima/maxima, which uses the one dimensional method of Brent [2]. In the next section a few experiments are demonstrated.

Problem with Powell's method

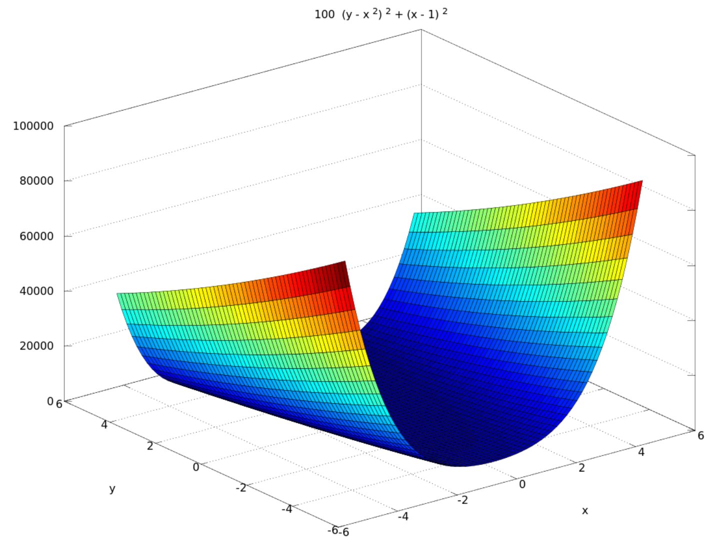
While Powell's method finds an extremum, it is often a local one. Powell's method requires an initial point from which it will start tracking the extremum and initial direction vectors in order to look for it in the direction that is respective for each variable. Different initial point and direction vectors will probably lead to different extremum. Below we have two examples of function minimization using Powell's method. In the first figure we have the Rosenbrock function, whose form is

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} [b(x_{i+1} - x_i^2)^2 + (x_i - a)^2],$$

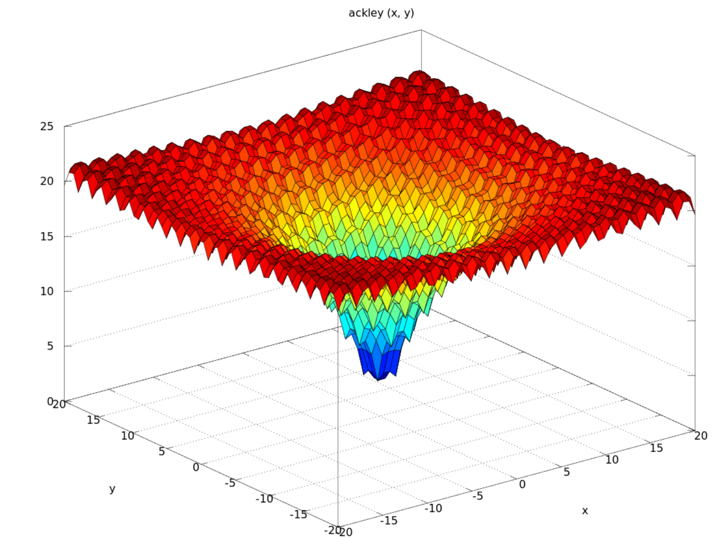
and the global minimum is at $(x, y) = (a, a)$ where $f(x, y) = 0$, and in the second figure we have the Ackley's function whose form is

$$f(x, y) = -a \exp\left(-b\sqrt{0.5(x^2 + y^2)}\right) - exp(0.5(\cos(cx) + \cos(cy))) + a + e$$

and the minimum is at $(x, y) = (0, 0)$ where $f(x, y) = 0$



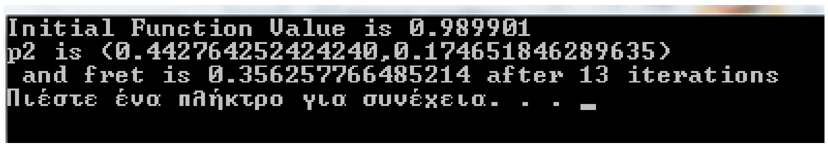
(a) Rosenbuck's function for N=2, a=1, b=100



(b) Ackley's function for a=20, b=0.2, c=2π

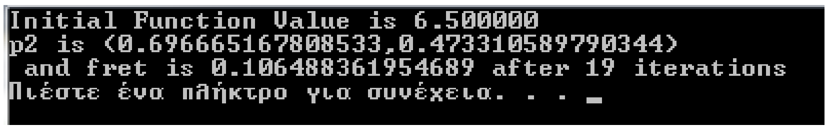
Figure 2: Test Functions for optimization

These functions are known test functions for optimization. Using Powell's method for optimization, we have the results below.



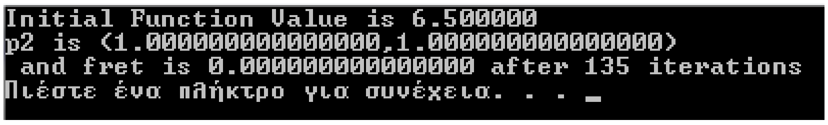
```
Initial Function Value is 0.989901
p2 is (0.442764252424240,0.174651846289635)
 and fret is 0.356257766485214 after 13 iterations
Πιέστε ένα πλήκτρο για συνέχεια. . . _
```

(a) Initial point:(0.01,0.01), Initial Direction vectors: (1,0),(0,1), local minimum



```
Initial Function Value is 6.500000
p2 is (0.696665167808533,0.473310589790344)
 and fret is 0.106488361954689 after 19 iterations
Πιέστε ένα πλήκτρο για συνέχεια. . . _
```

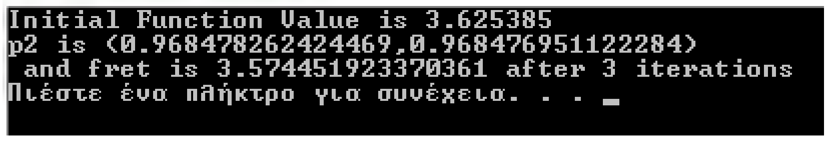
(b) Initial point:(0.5,0.5), Initial Direction vectors: (1,0),(0,1), local minimum



```
Initial Function Value is 6.500000
p2 is (1.000000000000000,1.000000000000000)
 and fret is 0.000000000000000 after 135 iterations
Πιέστε ένα πλήκτρο για συνέχεια. . . _
```

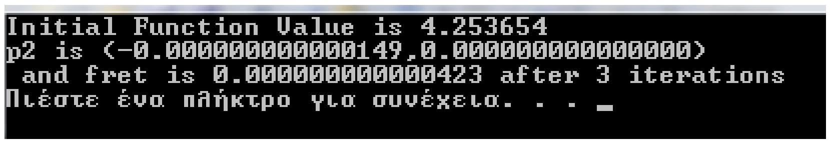
(c) Initial point:(0.5,0.5), Initial Direction vectors: (0,1),(1,0), Global minimum

Figure 3: Rosenbuck's function experiments



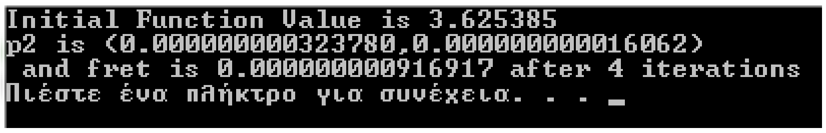
```
Initial Function Value is 3.625385
p2 is (0.968478262424469,0.968476951122284)
 and fret is 3.574451923370361 after 3 iterations
Πιέστε ένα πλήκτρο για συνέχεια. . .
```

(a) Initial point:(1,1), Initial Direction vectors: (1,0),(0,1), local minimum



```
Initial Function Value is 4.253654
p2 is (-0.000000000000149,0.000000000000000)
 and fret is 0.000000000000423 after 3 iterations
Πιέστε ένα πλήκτρο για συνέχεια. . .
```

(b) Initial point:(0.5,0.5), Initial Direction vectors: (1,0),(0,1), Global minimum

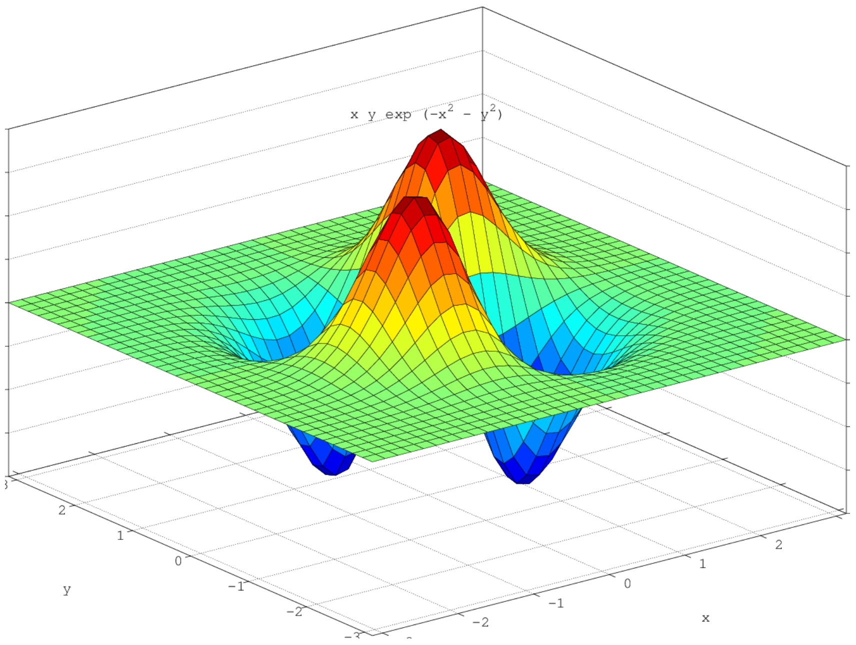


```
Initial Function Value is 3.625385
p2 is (0.000000000323780,0.000000000016062)
 and fret is 0.000000000916917 after 4 iterations
Πιέστε ένα πλήκτρο για συνέχεια. . .
```

(c) Initial point:(1,1), Initial Direction vectors: (10,0),(0,10), Global minimum

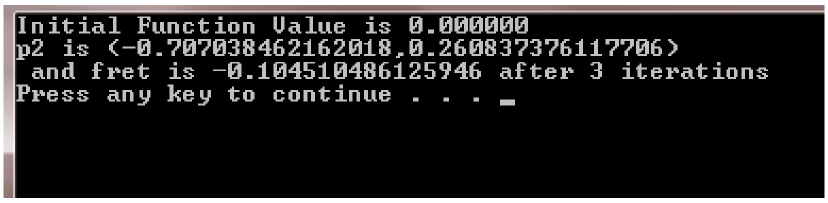
Figure 4: Ackley's function experiments

Another more simple example is this. Let $f(x,y) = xy\exp(-x^2 - y^2)$, whose graph for $x, y \in [-10, 10]$ is this:



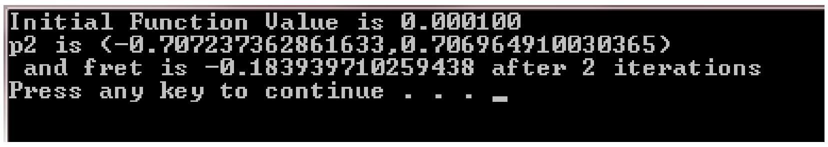
Its global minima are at $(x,y) = \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$ and $(x,y) = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ where $f\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right) =$

$\left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) = -\frac{1}{2e}$. Powell's method was used in ordered to see how well it responds to changes in initial point. In the first and the second experiment the initial point was $(0, 0)$ and $(0, 1)$ respectively and the initial direction vectors are $(0, 1), (1, 0)$. In the third experiment the intitial point is the same as that of the second experiment but the initial direction vectors are $(1, 0), (0, 1)$ Below we see the results of Powell's method.



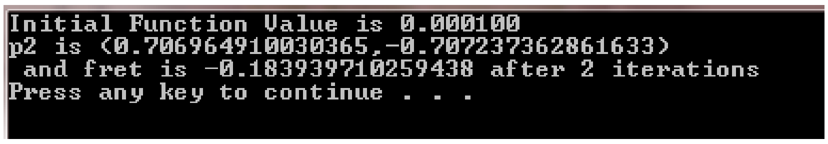
```
Initial Function Value is 0.000000
p2 is (-0.707038462162018,0.260837376117706)
 and fret is -0.104510486125946 after 3 iterations
Press any key to continue . . .
```

(a) Initial point:(0,0), Initial Direction vectors: (0,1),(1,0)



```
Initial Function Value is 0.000100
p2 is (-0.707237362861633,0.706964910030365)
 and fret is -0.183939710259438 after 2 iterations
Press any key to continue . . .
```

(b) Initial point:(0.1,0.1), Initial Direction vectors: (0,1),(1,0)



```
Initial Function Value is 0.000100
p2 is (0.706964910030365,-0.707237362861633)
 and fret is -0.183939710259438 after 2 iterations
Press any key to continue . . .
```

(c) Initial point:(0.1,0.1), Initial Direction vectors: (1,0),(0,1)

Figure 5: Experiments of minimizing $f(x, y) = xy \exp(-x^2 - y^2)$

In the first experiment the result was a local minimum, but a slight change of the initial point in the second experiment leads to the location of the global one. Finally, in the third experiment the reverse of the initial direction vectors leads to the location of the other global minimum. The choice of good initial point and direction vectors becomes a dilemma as the dimension of the function increases and the function is not regular.

Application of Maes' Method and Experiments

In the previous section the problem of Powell's method was discussed. Here the same problem will be viewed as Powell's optimization method is used in Maes' method. First, the form of the Maes' mutual information as function will be studied and the simplest way to do it is to see how the mutual information between an image and a rotated/ translated version of the former changes as the angle/translation changes. Below we have three graphs of Mutual information vs rotation, translation in x and y axis respectively.

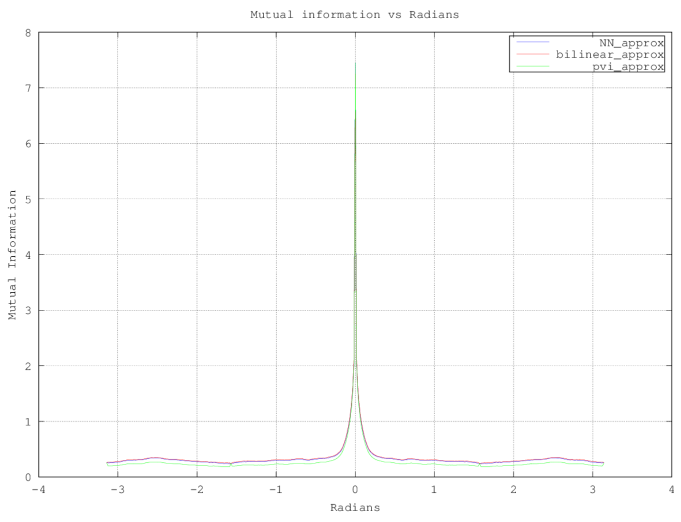


Figure 6: Mutual Information vs Radians

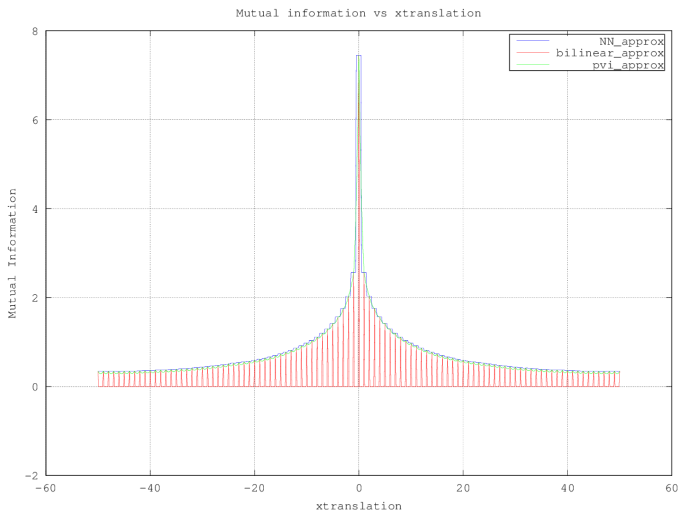


Figure 7: Mutual Information vs translation along x axis

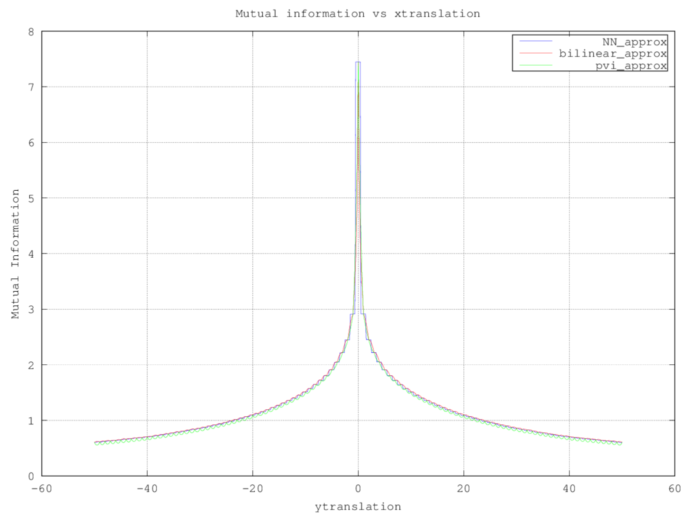
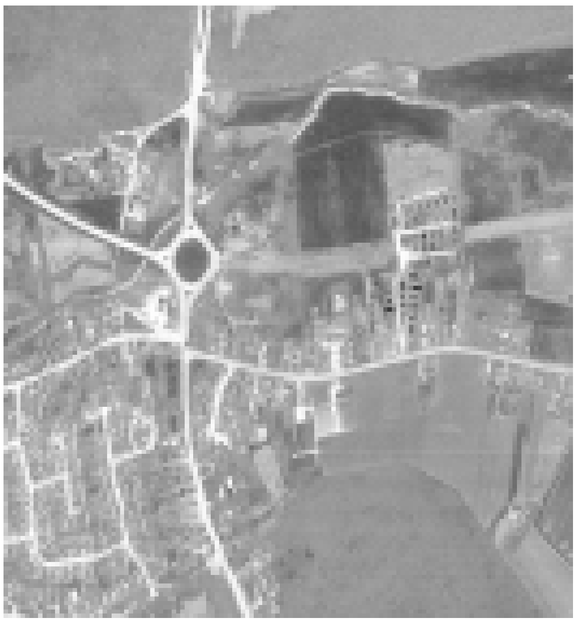
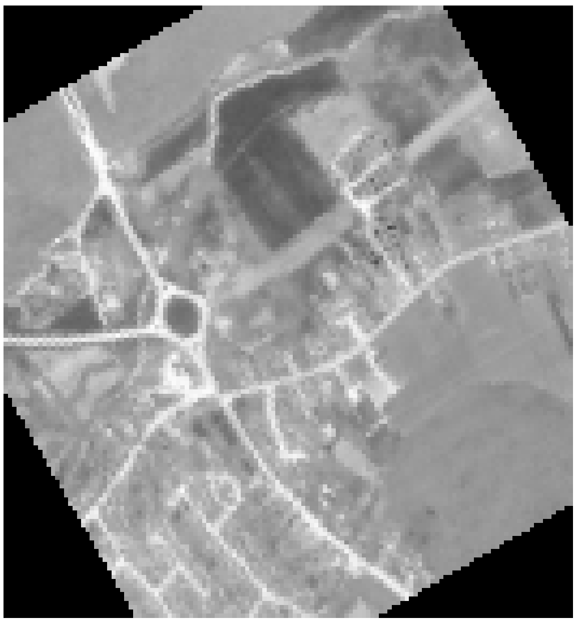
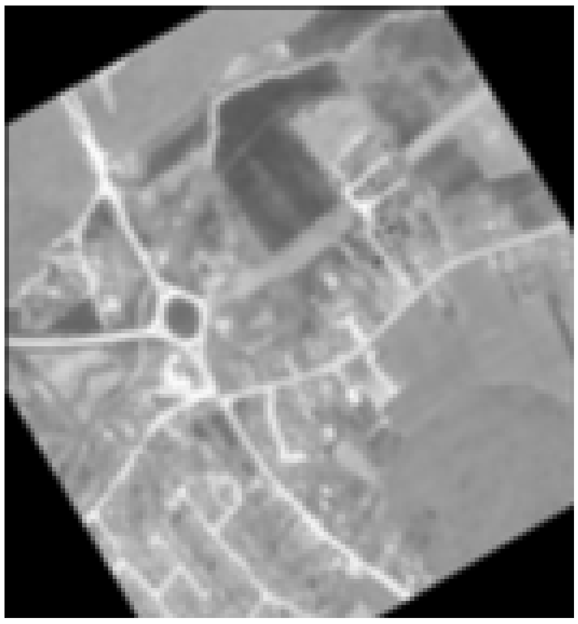


Figure 8: Mutual Information vs translation along y axis

The graphs show that, no matter which interpolation method is used, mutual information as a function tends to have many local maxima, therefore the initial point and direction vectors must be carefully chosen. In this section the results of a few indicatice experiments are presented, in order to have a clearer sight of the pitfalls Maes' method stumbles on. The first three figures show successful image registration.(The first one is from this site: `http://www.robots.ox.ac.uk/~cvrg/trinity2002/seb/results.html`)


(a) Original Image of Airport


(b) Target Image of Airport


(c) Transformed original image

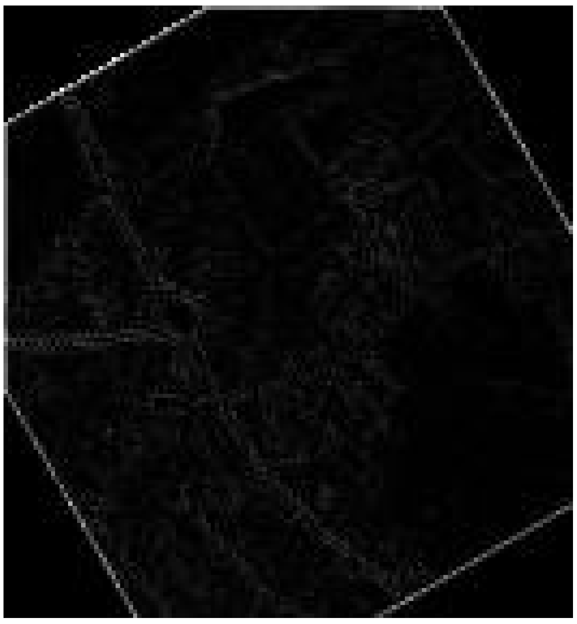
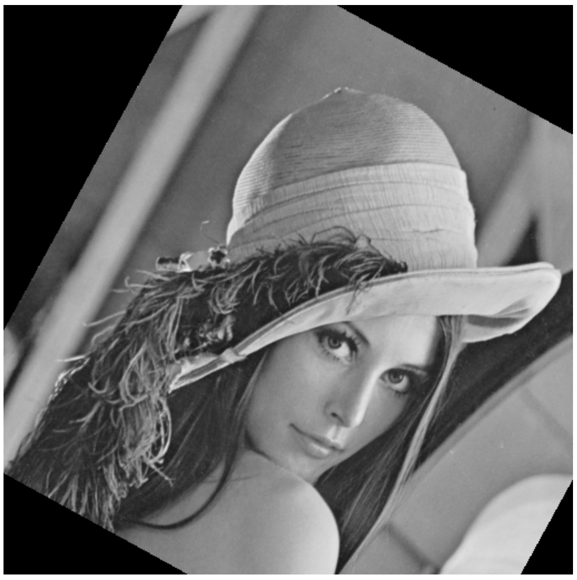
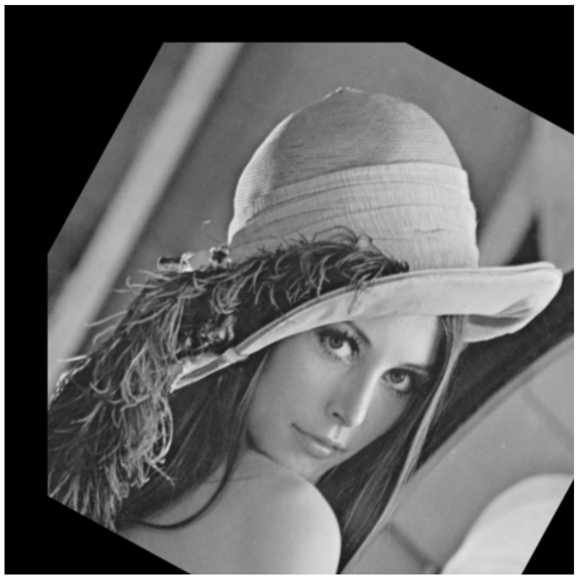

(d) Comparison of (b) and (c)

Figure 9: Synthetic Experiment: Image Registration of Airport

(a) Original picture of Lena


(b) Rotated and Translated


(c) Transformed original image

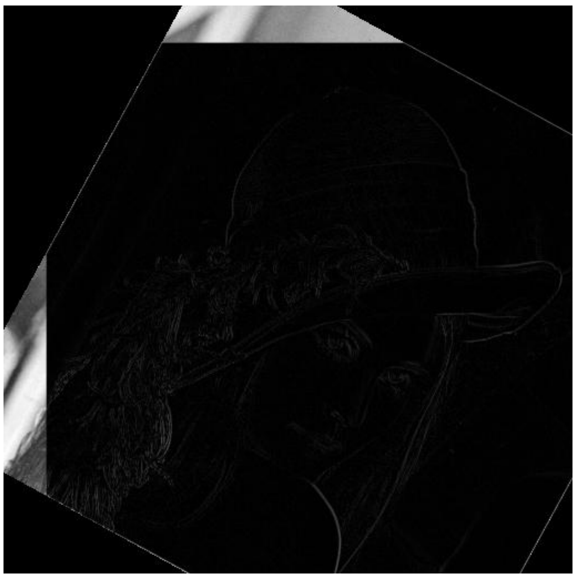
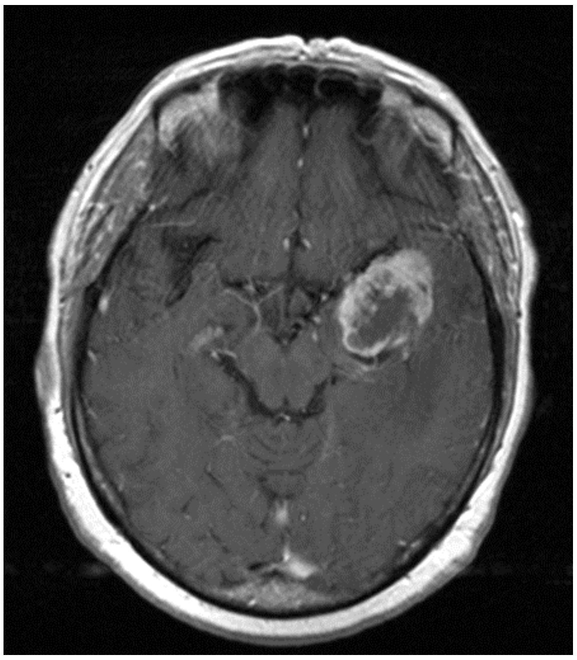
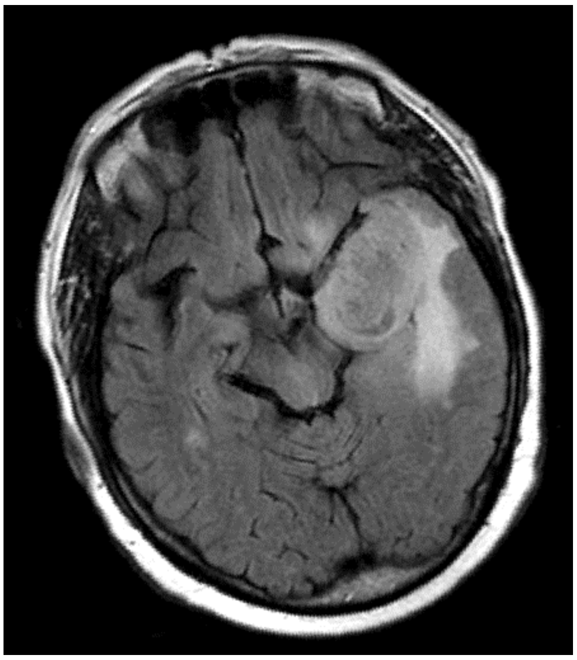
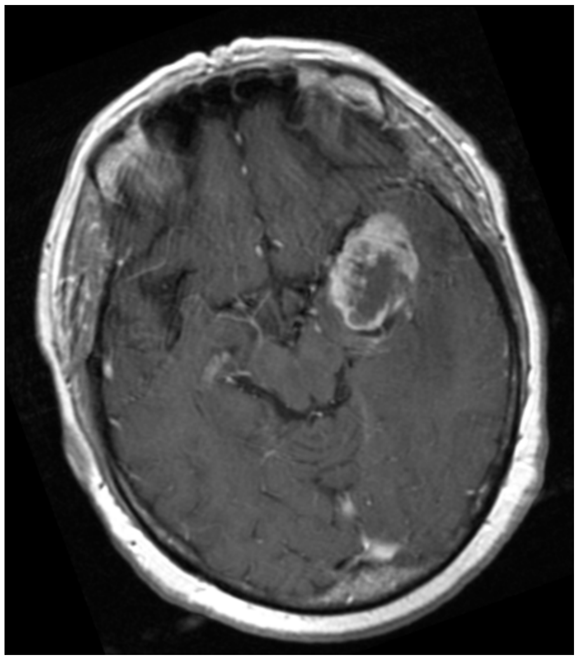

(d) Evaluation of registration

Figure 10: Synthetic Experiment: Image Registration of Lena's pictures


(a) MRT1


(b) MRT2


(c) Transformed MRT1

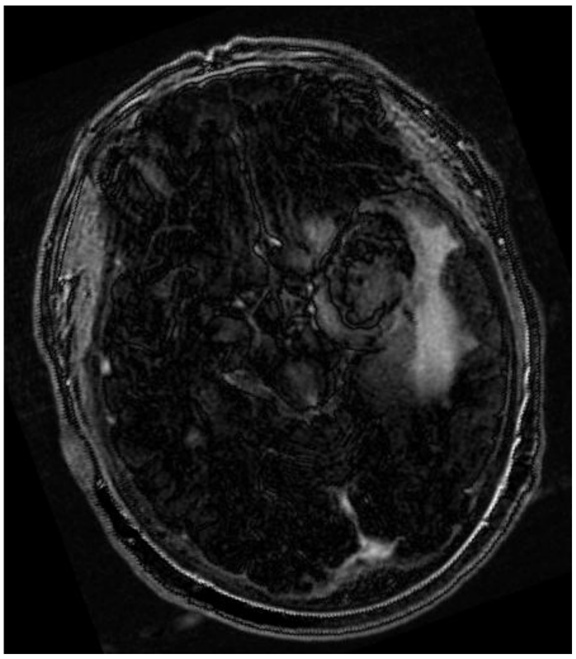
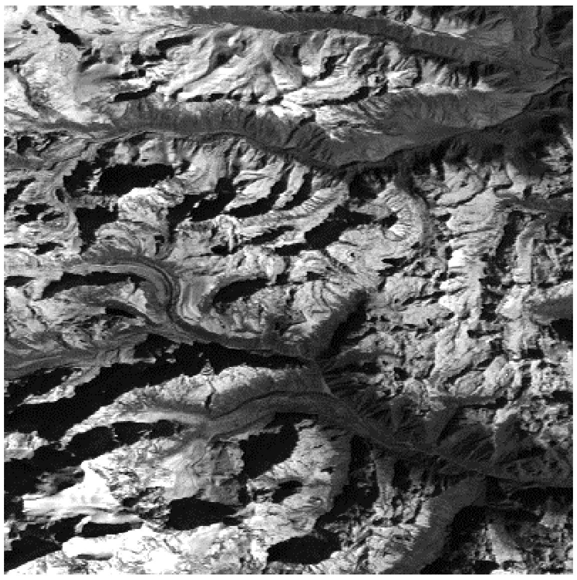
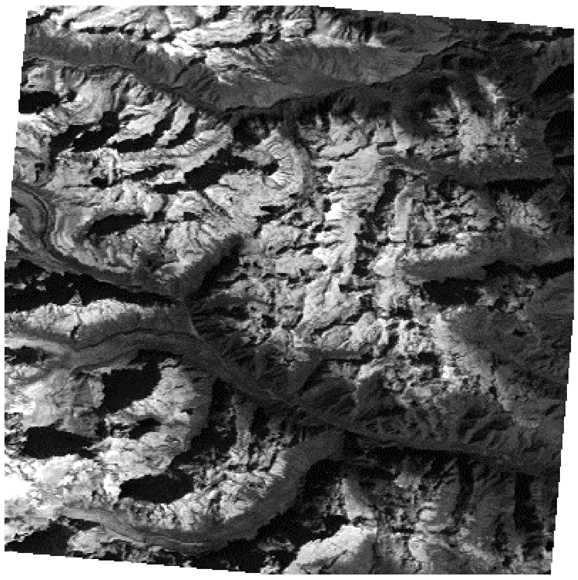
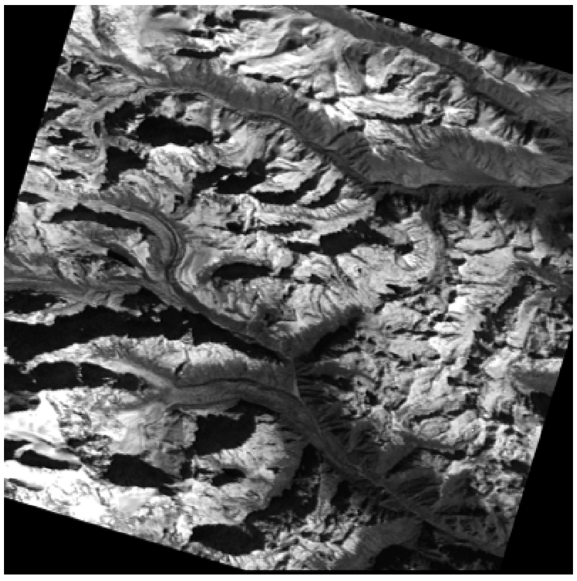

(d) Comparison of (b) and (c)

Figure 11: Image Registration of T1 and T2 scans

Now we show pairs of images for which the method of Maes didn't produce good results.


(a) Floating Image


(b) Reference Image


(c) Failed Registration

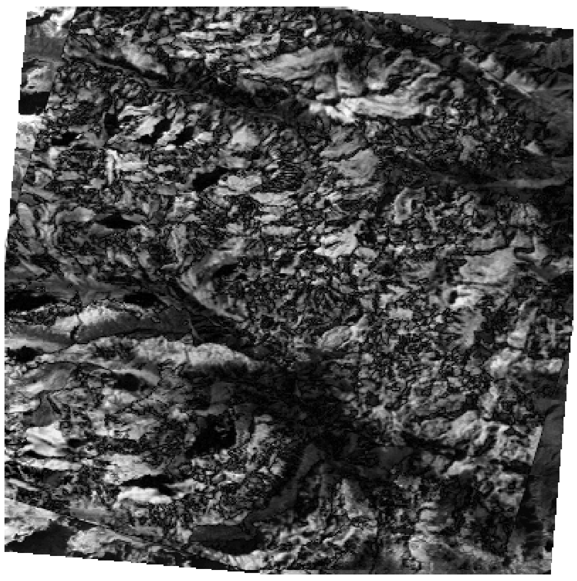
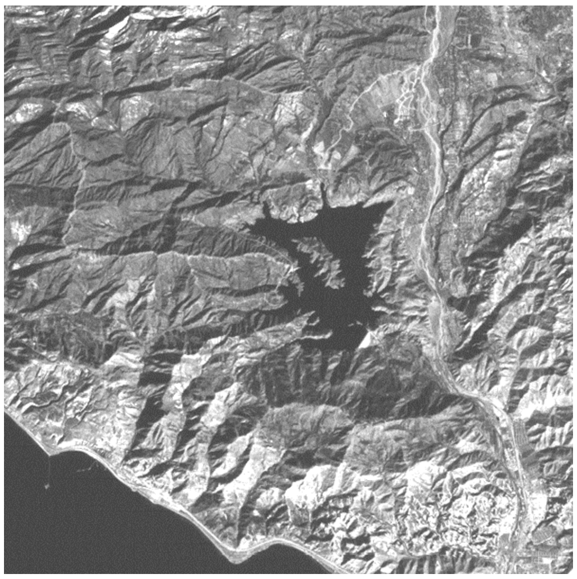
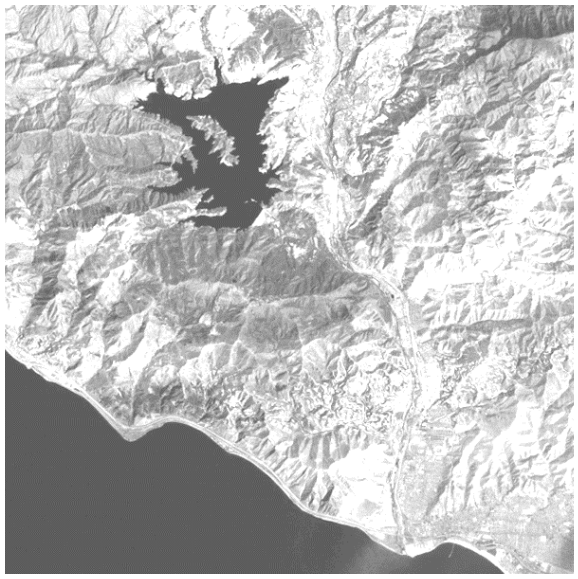

(d) Comparison of (b) and (c)

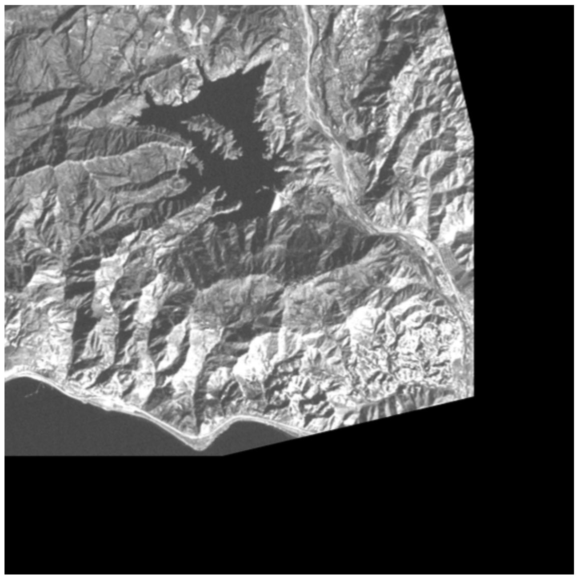
Figure 12: Mountainous Area Image Registration



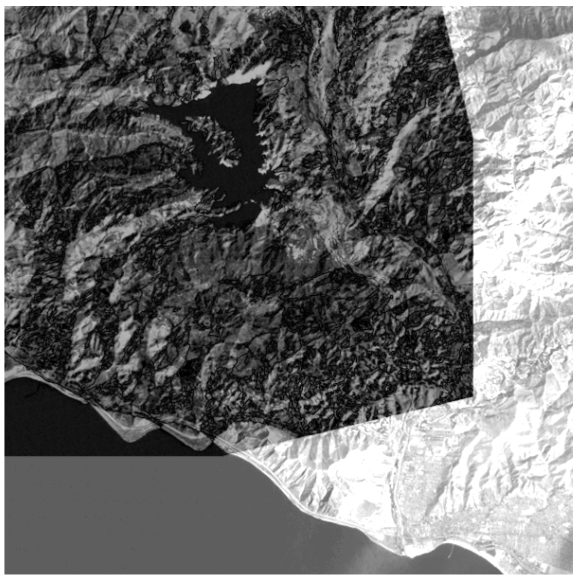
(a) Floating Image



(b) Regerence Image

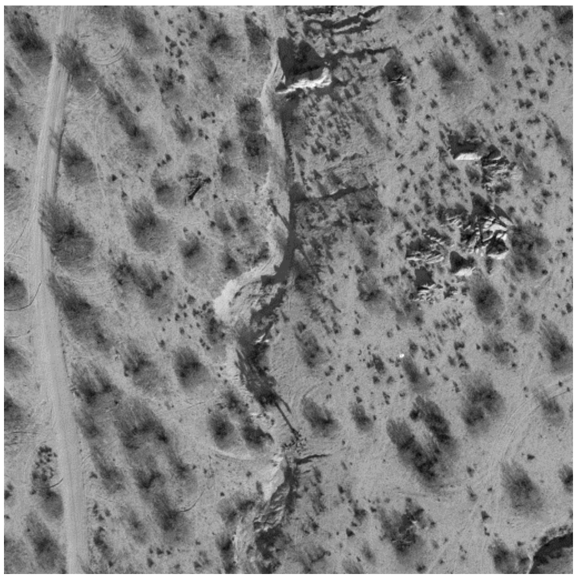


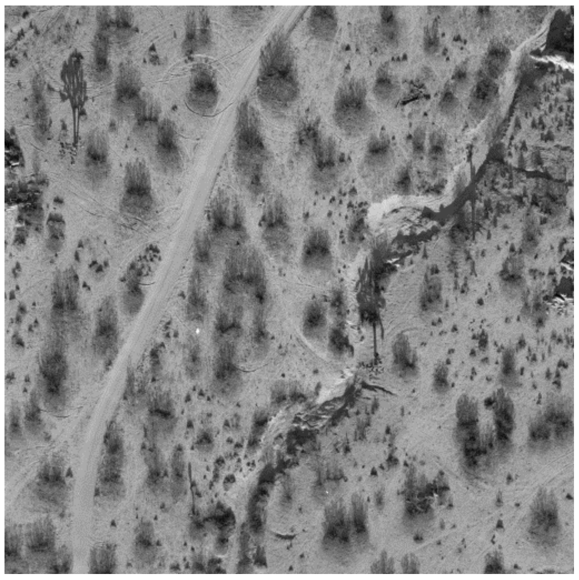
(c) Failed registration

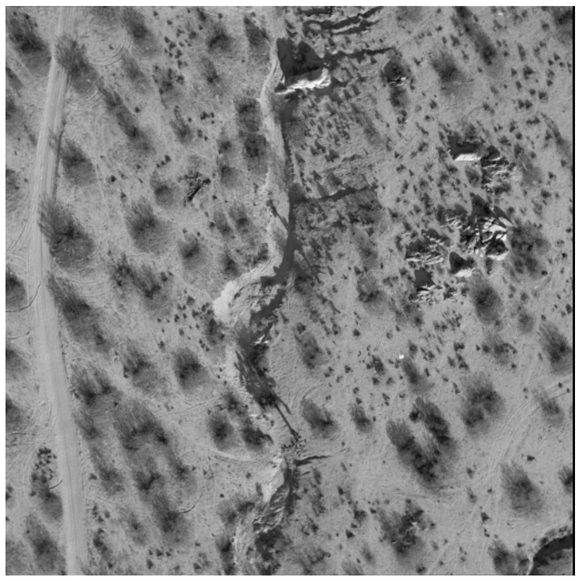


(d) Comparison of (b) and (c)

Figure 13: Landscape multi-modal image Registration


(a) Floating Image


(b) Reference Image


(c) Failed registration

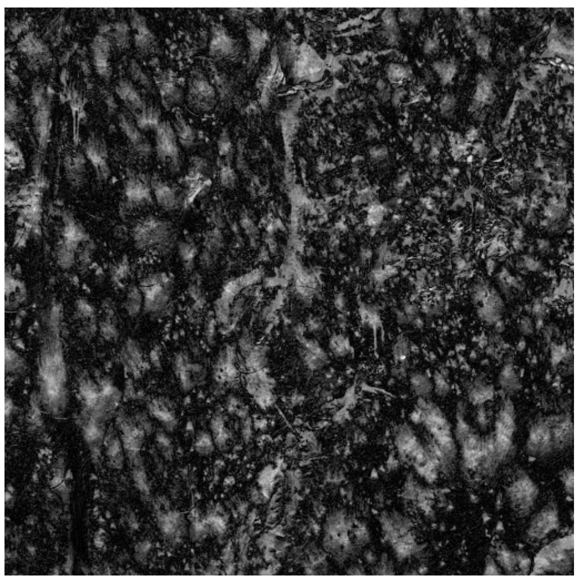

(d) Comparison of (b) and (c)

Figure 14: Landscape image registration

In all the experiments above the initial transform was the identity transform, i.e

$$Ts = Rs + t,\ \ R = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ \ t = [0,0]^T$$

and the initial direction vectors were the unit vectors in each parameter direction. The method seems to give better results, when we are dealing with images that have relatively simple patterns, that can be segmented into dissimilar smaller areas and without many symmetries. Images of bad quality, with very complex patterns tend to be a difficult case for Maes' method. The reason is that in the successful experimetns, Maes' mutual information tends to have relatively few maxima, therefore the initial transform and the direction vectors don't play such an important role as in the case of the other examples. The images of mountain areas and the first Lanscape have many complex patterns and there are many symmetries in the first landscape. That means that we can have many maxima and the choice of a wrong initial transform and direction vectors, tend to give very bad results. In order to understand this better, the graphs showing the steps of the algorithms below are presented. The graphs are based on registration of the first failing example of Maes' method, using the three strategies proposed by Maes.

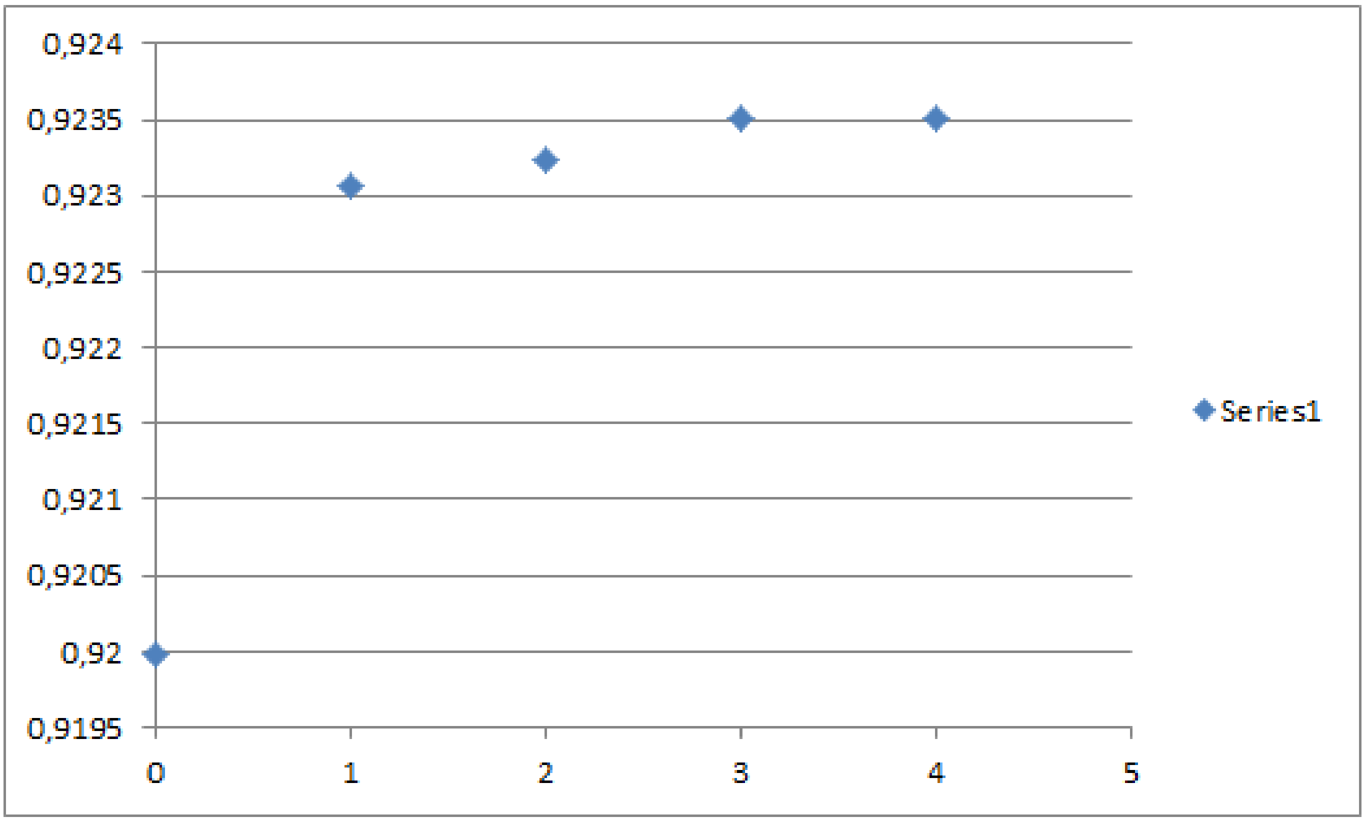


Figure 15: Nearest Neighbour approximation

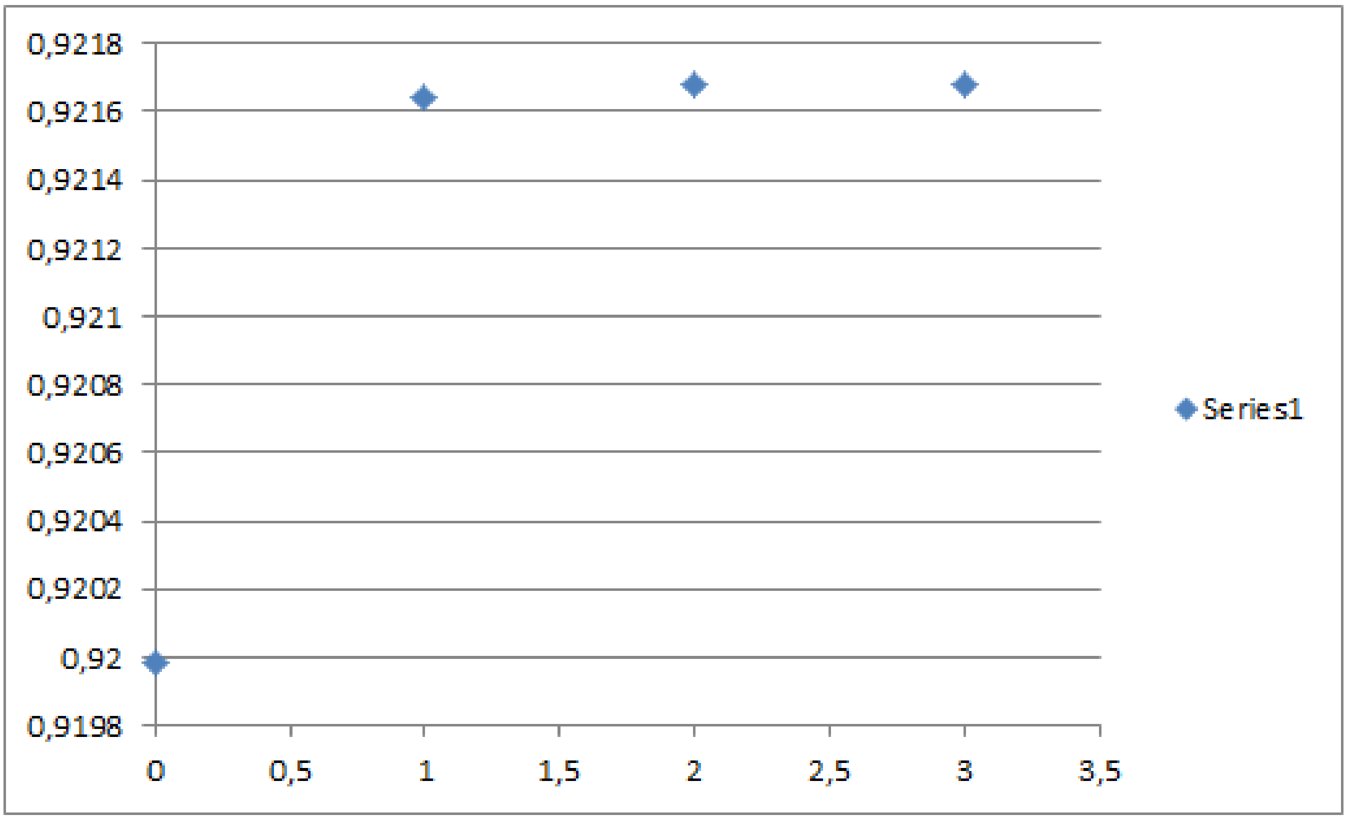


Figure 16: Bilinear Interpolation approximation

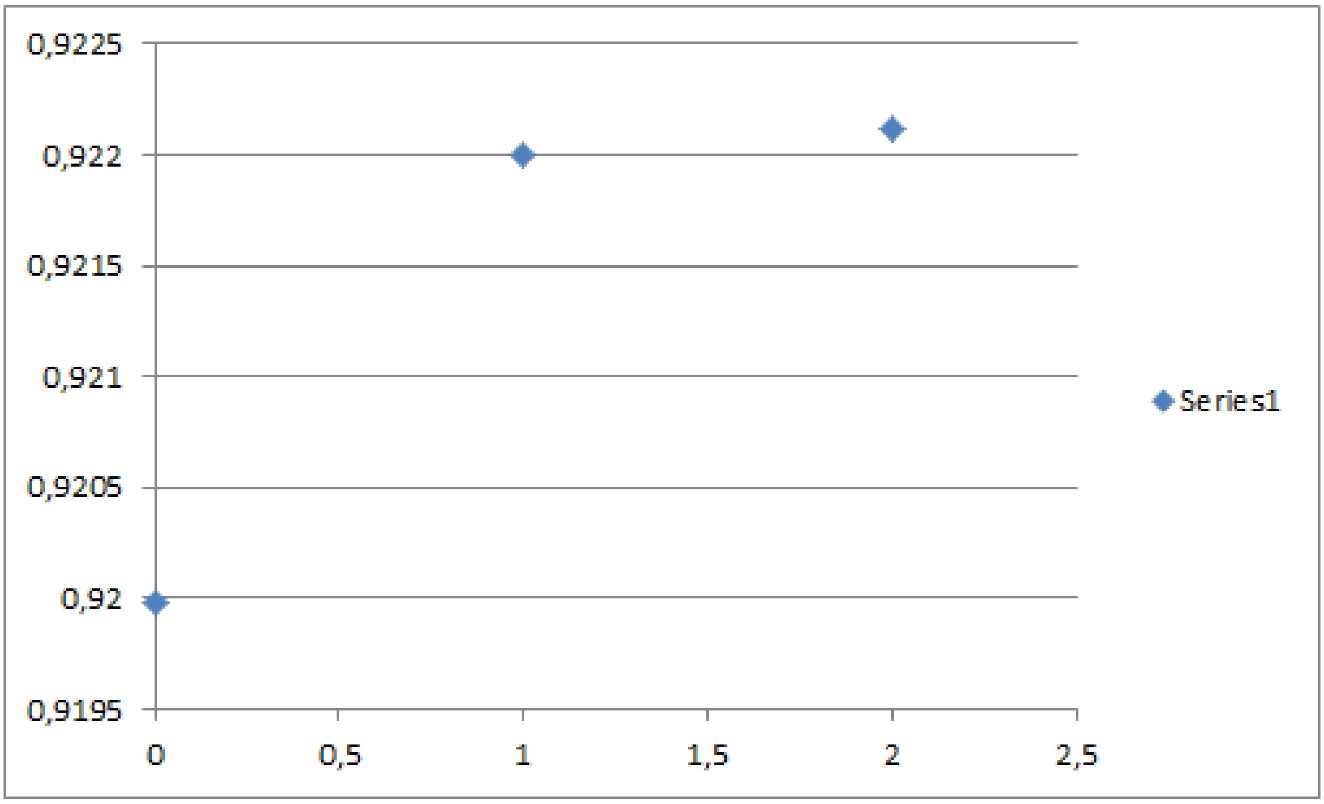


Figure 17: PVI approximation

In the graphs above, we see the plots of Mutual Information vs iteration for each mentioned strategy. The initial transform and the initial direction vectors are the same parameters that were used in previous experiments. As we see, Mutual Information barely increased, indicating that the final transform matrix is barely different from the initial transform matrix. After all, powell's method finds a local extremum, which may not coincide with a global one. In the next graph we see the mutual information computed the extended method in every iteration.

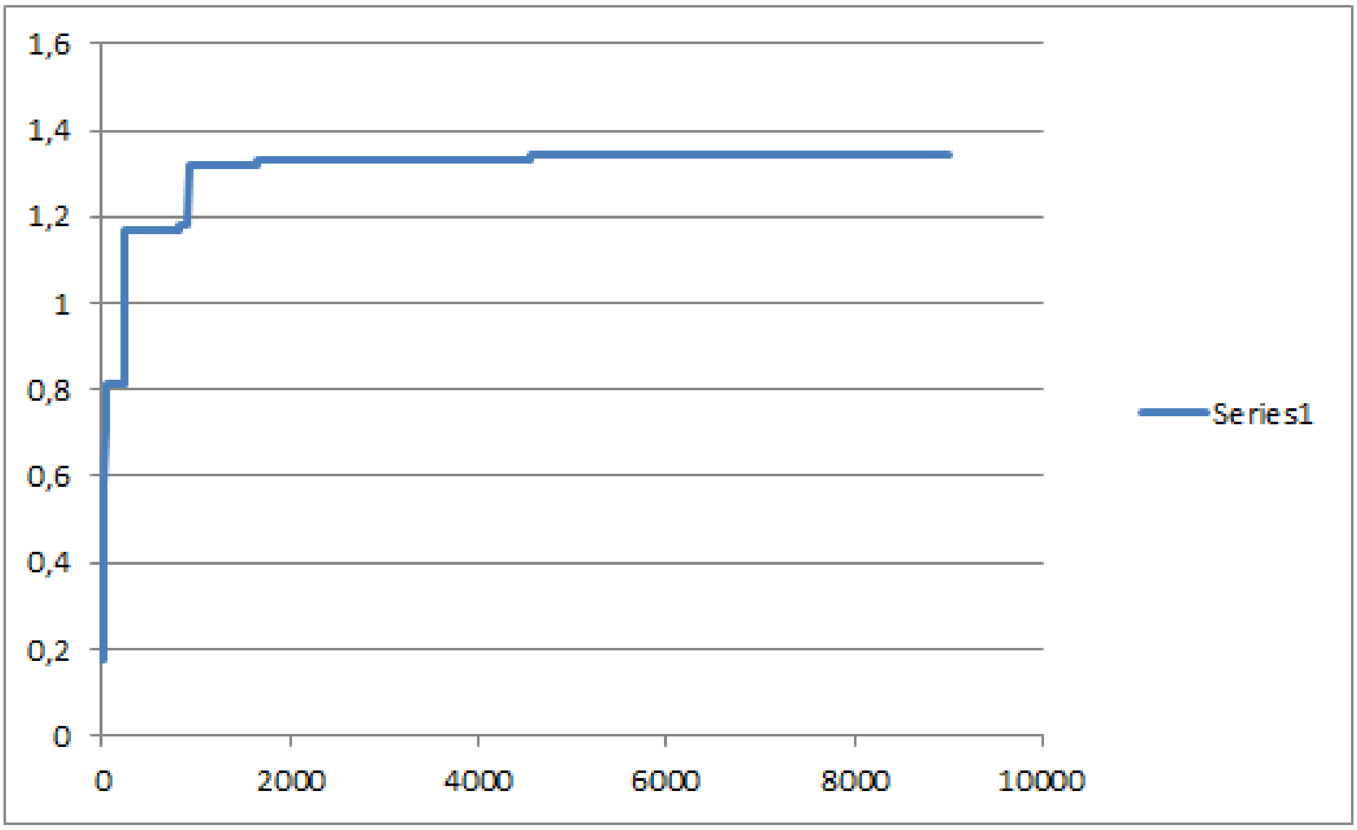
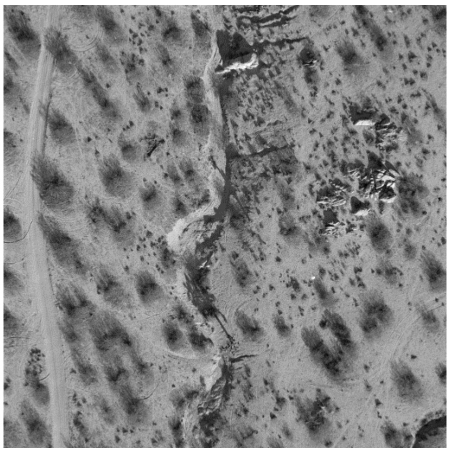


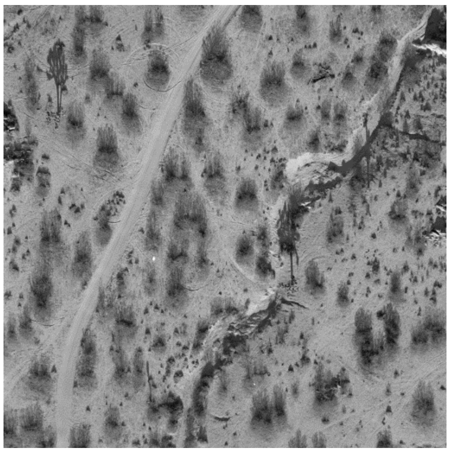
Figure 18: Mutual information vs iteration

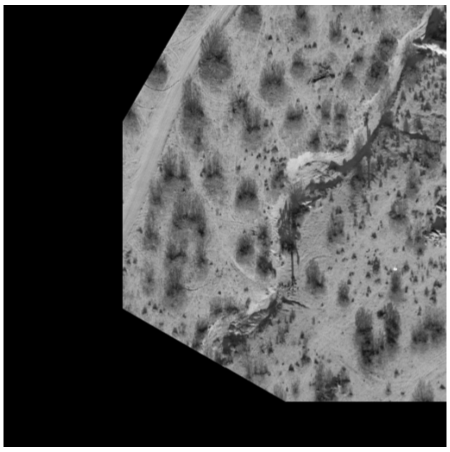
As we see, the algorithm won't stop at a local extremum and will search for the global one until the termination criteria are met. The correct transform is

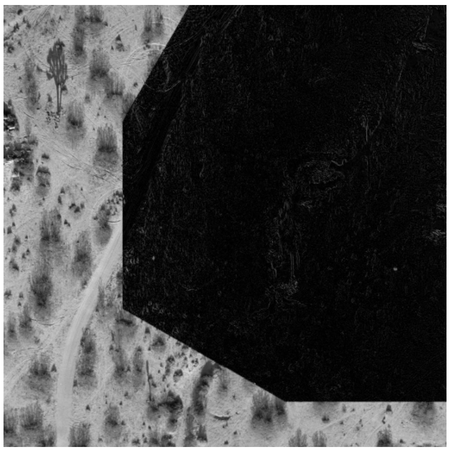
$$(Angle,\ X,\ Y) = (0.518, 138, -52.6)$$

and the following images are the results of the registration.


(a) Source Image


(b) Target Image


(c) Transformed Source Image


(d) Difference Image of (c) and (b)

In the next section, the extended method is presented, explaining how it bypasses the restrictions of Maes' method.

Extended Method

As we've seen in the previous section, Maes' method has some limitations regarding its ability to cope with images of bad quality or with complex patterns or/and symmetries. In this new approach, Maes' mutual information as a similarity measure is kept. The search is done with a different way, though.

Genetic Algorithm

Genetic algorithms [13] is a heuristic search that emulates natural selection. In natural selection it is stated that, given a population of a species, its members reproduce, passing their characteristics to their descendants. Along with breeding, there is also a chance of mutation,

which leads to new characteristics, either better or worse. The better characteristics tend to become dominant in the population and the bad ones gradually disappear.
In genetic algorithm, a population of candidate "solutions" is evolved to better solutions. We start with an initial random population, of which each candidate solution has a mutating set of properties. In every iteration, a few selected candidates of the current population create a new population, called generation, of which every member's fitness is evaluated. The fitness is the value of the objective function of the optimization problem (here is the mutual information). The more fit members tend to be more often selected from the current population (without totally excluding candidates of poor fitness)and their genome is modified though recombination and mutation. Thus a new generation is created that is used in the next iteration to form another one. The algorithm terminates when the termination criteria are satisfied. More analytically, the general form of a genetic algorithm is this:

```
function G.A($obj\_fun$,$m\_rate$,$c\_rate$)
    $Old\_Population \leftarrow Initialize()$
    while true do
        $Selected\_Members = Selection(Old\_Population)$
        $New\_Population \leftarrow Crossbreed(Selected\_Members, c\_rate)$
        $New\_Population \leftarrow Mutate(New\_Population, m\_rate)$
        if criteria met then
            break
        end if
        $Old\_Population \leftarrow New\_Population$
    end while
    $return\ best\ from\ New\_Population$
end function
```

The arguments of the algorithm *obj_fun*, *m_rate*, *c_rate* are the objective function (in our case the mutual information), the mutation rate and the crossover rate respectively. The crossover rate is a parameter that shows how big is the possibility for the current population members to mix their characteristics to produce new candidate solutions. The higher the rate, the greater the variety of the new solutions. The mutation rate is a parameter that shows how easily the candidate solutions mutate. Its job is to prevent the algorithm from getting to a local extremum without any possibility of finding a better solution. Especially in problems where the objective function has many extrema, a high mutation rate is needed.
The termination criteria of the algorithm are the following:

- Maximum number of generations is reached
- Manual inspection
- The highest ranking solution's fitness is reaching or has reached a plateau such that any

further search is pointless since no better results can be produced

- A combination of the above

Selection

Selection is the process through which a fraction of candidate solutions' population is chosen for reproduction in order to create a new population of candidate solutions. In biology, natural selection tends to favour those that are fit to survive, giving them the advantage to survive and have more offsprings. Depending on the environmental conditions, the fitness of an organism can increase or decrease. There is also the fact that even the organisms with poor fitness may have good genes. That means that candidates with fairly good fitness are favoured without (totally) excluding the ones with poor fitness. The selection, as it is done in nature, is simulated in genetic algorithms. The procedure is to select the a number of candidates for reproduction whose majority (most probably) has good fitness, with a small minority having a poorer one. A few selected methods for selection are presented below.

Truncation Selection

Truncation Selection [11] is the simplest and probably the crudest method. The idea is to order the population with respect to fitness in ascending order and take a portion p of the fittest candidates. These candidates will reproduce 1/p times. This method totally excludes the candidates with poor fitness (even those that might have good genes). Because of its crudeness, it's not often used in practice.

Tournament Selection

Tournament Selection [9, 10] runs several "tournaments" among k random individuals of the current population, the winner of each one is to reproduce. The number k of the random individuals is crucial for the selection pressure. The larger the k, the smaller the chance for a weak individual to be chosen. The pseudo-code is this:

```
function TS($population,k,p$)
    $constetants \leftarrow random\ k\ from\ population$
    $choose\ best\ with\ probability\ p$
    $choose\ second\ best\ with\ probability\ p(1-p)$
    $choose\ third\ best\ with\ probability\ p(1-p)^2$
    $\vdots$
    and so on
end function
```

The method's advantages are its simplicity, ability to work on parallel architectures and the adaptable selection pressure.

Fitness Proportionate Selection

Fitness Proportionate Selection [8] is another method of choosing candidates for reproduction. Each candidate i has a fitness value f_i. Therefore its probability of selection is $p_i = \frac{f_i}{\sum_{j=1}^{N}} f_j$, where N is the population size. This selection process looks like a roulette in a casino where each candidate is given a portion of the wheel according to its fitness, with the fittest ones having the bigger portions. By "turning the wheel" there is a small chance to choose a less fit solution and eliminate a good one, which isn't necessarily bad, because a less fit solution may still have good genes, therefore making it more preferable than the truncation method.

Stochastic universal sampling

Stochastic universal sampling [12] is an advanced form of fitness proportionate selection that is able to deal with bias and minimal spread. Instead of choosing candidates for production by using repeated random sampling, like fitness proportionate selection does, stochastic universal sampling uses a single random value to sample the candidates by choosing them at evenly spaced intervals, therefore, giving the weaker candidates a chance to be chosen and reduce the unfairness between weak and strong candidates that exists in fitness proportionate selection. The pseudo-code for the method is this:

```
function SUS($population$,$N$)
    $F \leftarrow$ TOTAL FITNESS
    $N \leftarrow$ NUMBER OF OFFSPRING TO KEEP
    $P \leftarrow$ DISTANCE BETWEEN THE POINTERS (F/N)
    $Pointers \leftarrow vector(N)$;
    for $i = 0 \to N - 1$ do
        $Pointers[i] \leftarrow Start + i * P$
    end for
return newPopulation(population,Pointers)
end function
function NEWPOPULATION($Population$,$Points$)
    $Chosen \leftarrow []$
    $i \leftarrow 0$
    for $P$ in $Points$ do
        while FITNESS SUM OF POPULATION[1 I] $< P$ do
            $i++$
        end while
        $Apprend(Population[i], Chosen)$
    end for
return Chosen
end function
```

Crossover

Crossover is the process through which candidate solutions with (possibly) good fitness combine their ”genomes” so that new candidate solutions are produced with presumably better fitness. In biology, when two parents reproduce, the offsprings have a genome which is a combination of that of their parents. In genetic algorithms, crossover is an exchange in genes between the candidate solutions. Here are a few characteristic methods of Crossover.

One point crossover

The candidates' genome is spliced in two sub-genomes on one point which is the same for them. Then one part of the first candidate is exchanged for the respective one of the other. A generalization of this method is k-point crossover, where the genomes are split using k points where the first respective part are swapped, the second remain unchanged, the third are swapped, etc.

Cut and splice

The genomes are spliced at a separate point in every candidate. Then an exchange leads to different length.

Uniform Crossover and Half Uniform Crossover

These methods create new candidates from existing ones by swapping the latters' genome in gene level. That means that the parents contribute the gene and not whole segments of their genome. The bits of the parents are compared between them and then swapped with a fixed probability (usually 0.5). In half uniform crossover, only half of the different bits are swapped.

Mutation

Mutation is the key to evolution through which organisms can have a benefit over other competitors or go extinct. The same idea is applied in genetic algorithms. After every reproduction that gives a new generation, a series of mutations occur and the genome of the candidate solutions is altered. The reason for this is that there might be a probability that the best candidate of the new generation is (only) locally the best solution to our optimization problem. Mutation provides further diversification, which prevents stumbling upon locally best solutions by allowing the exploration of further solutions. Without mutation, the members of the population eventually would become too similar and the process of evolution would be stagnant. Since there are many genome types, there are many mutation methods, each corresponding to specific genome type(s).

Bit string mutation

The mutation in this method is done by flipping random bits of the string. The probability of mutation of a bit is $\frac{1}{l}$ where l is the length of the bit vector.

Flip Bit

In this method, the bits of the string are flipped, e.g 0110100 becomes 1001011.

Boundary

If the genome is integer or float, then the gene is replaced by the upper or the lower boundary value of it.

Non-Uniform

Used only for integer and float types of genes, this method ensures that, as the number of generation increases, the probability of mutation will converge to 0. This method boosts the diversity of candidates in the early stages of the genetic algorithm in order to prevent stagnation and as the number of generation increases the algorithm converges to (possibly) the global extremum.

Uniform

The gene is replaced by a uniform random number that is selected between its lower and upper boundary value.

Gaussian

With this method a new Gaussian distributed random value is added to a gene. If the sum is within the boundary values of the gene, then the gene retains its new value, or else the new value is clipped.

Elitism

Elitism is a variant of the general process of new population construction. There is always a chance that the new population's best candidate may be worse than the best candidate of the previous population, therefore it may be useful to retain the best candidate of the previous generation unchanged in the new one. The idea is compare the previous and the new best candidate. If the new is better than the previous, then the previous best candidate is discarded. If the opposite happens, then the new best is remains in the population replacing one of the existing candidates (commonly the worst). In this way, the previous best candidate may pass into the next generation unchanged without the need to evolve further, which may help us in preventing the descend from a (possibly) global extremum to local one. This variant is applied in the extended method.

Experiments

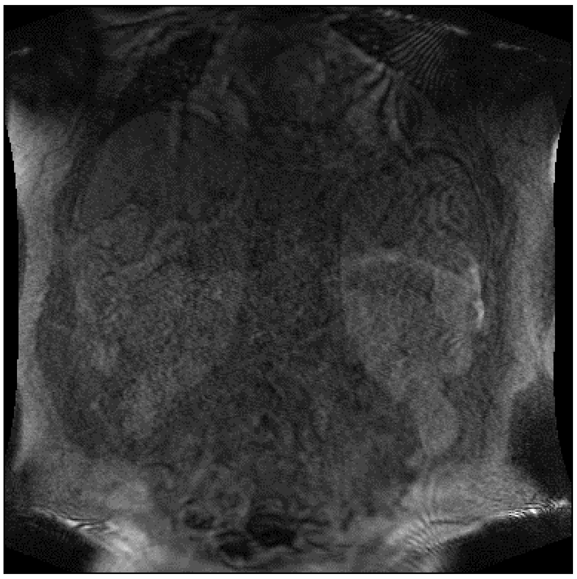
In this section we shall see the results of the experiments when we use the extended method, along with evaluations of the extended method in comparison with ITK's methods (first method uses Normalised Mutual Information with One Plus One evolutionary optimizer while the second uses Mates' Mutual information with regular step gradient descent optimizer) by using Feature Similarity Index [5], Structural Similarity Index [6] and the later's variant, Complex Wavelet Structural Similarity Index [7]. For the experiments, the mutation rate, the crossover rate, the number of generations and the population size were set at 0.85, 0.95, 20000, 100 respectively. The images are chosen so that there is diversity in photometric conditions, modality, with the goal of pushing the image registration methods to their limits. The following table shows the evaluation of the extended method. The maximum value for each evaluation of the indices above is 1. The higher the evaluation is, the grater the similarity between the images. The images came from the following image databases:

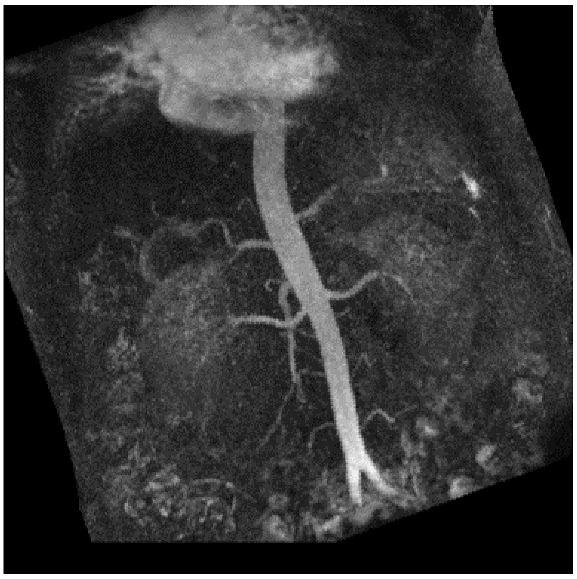
- `http://www.insight-journal.org/rire/`
- `http://vision.ece.ucsb.edu/registration/satellite/testimag/index.htm`
- `http://www.physionet.org/physiobank/database/images/`
- `http://www.radiologyinfo.org/en/photocat/gallery2.cfm?pg=angioct`

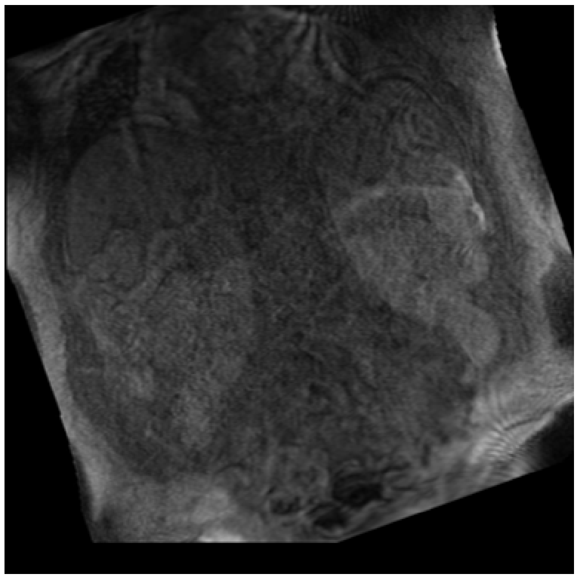
Evaluations				
Images	Method	FeatSIM	SSIM	CWSSIM
MRT1→Angiogram	Extendend	0.7324	0.4075	0.3444
	NMI	0.6685	0.2756	0.3703
	MatesMI	0.7115	0.2807	0.3329
b040→b042	Extendend	0.8483	0.7071	0.7218
	NMI	0.6748	0.1369	0.4452
	MatesMI	0.6742	0.1352	0.4583
casitas84→casitas86	Extendend	0.7985	0.6322	0.8535
	NMI	0.6294	0.1179	0.5190
	MatesMI	0.5896	0.1058	0.4210
CT→MRT1	Extendend	0.7543	0.6516	0.7930
	NMI	0.7316	0.6333	0.7940
	MatesMI	0.7339	0.6354	0.7916
gibralt84→gibralt86	Extendend	0.8130	0.6181	0.8330
	NMI	0.8319	0.6903	0.8333
	MatesMI	0.6441	0.1779	0.5315
mono1→mono3	Extendend	0.8683	0.7755	0.9905
	NMI	0.5742	0.3317	0.2554
	MatesMI	0.5721	0.2953	0.2619
mrpd→mrt1	Extendend	0.8674	0.8229	0.8630
	NMI	0.8642	0.8152	0.8593
	MatesMI	0.8624	0.8134	0.8594
mrpd→mrt2	Extendend	0.8017	0.5793	0.7401
	NMI	0.7953	0.5542	0.7394
	MatesMI	0.7956	0.5545	0.7397
mrt1→mrt2	Extendend	0.7707	0.5372	0.6068
	NMI	0.7633	0.5068	0.6032
	MatesMI	0.7625	0.5113	0.6036
mtn1→mtn3	Extendend	0.8406	0.7870	0.9572
	NMI	0.5714	0.0729	0.5676
	MatesMI	0.5745	0.0630	0.5285
inner→synthetic inner	Extendend	0.8122	0.4639	0.9657
	NMI	0.8073	0.4847	0.9337
	MatesMI	0.5600	0.0781	0.4995

Table 1: Evaluation of registration technincs

Here the results of registration experiments using the extended method are presented. In each figure, the subfigure (a) is the Floating image, (b) the Reference image, (c) is the Transformed Floating image using the optimal transform that is found using the extended method and (d) is the comparison of subfigures (b) and (c). Each figure of registered image pair is followed by a figure of diagrams showing the search of the optimal transformation by the extended and the two methods of ITK mentioned above.


(a) Floating Image


(b) Reference Image


(c) Successful Registration

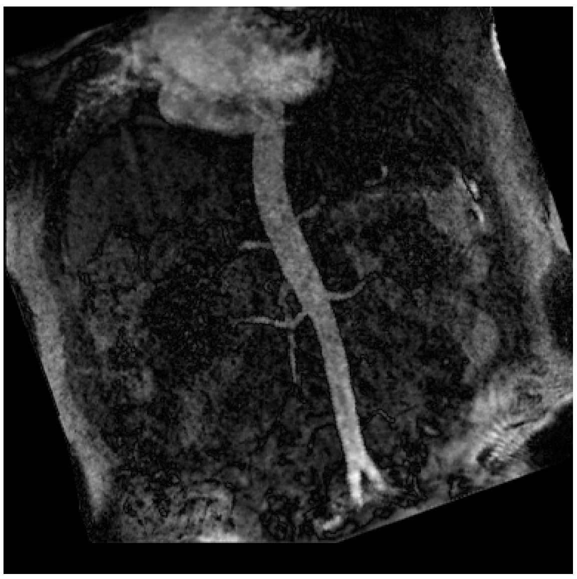
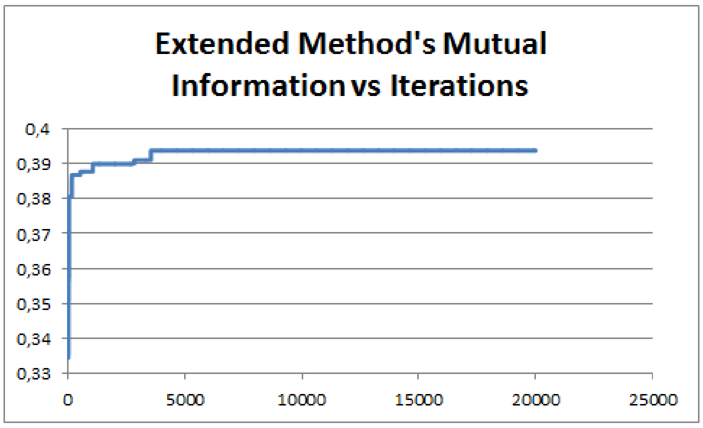
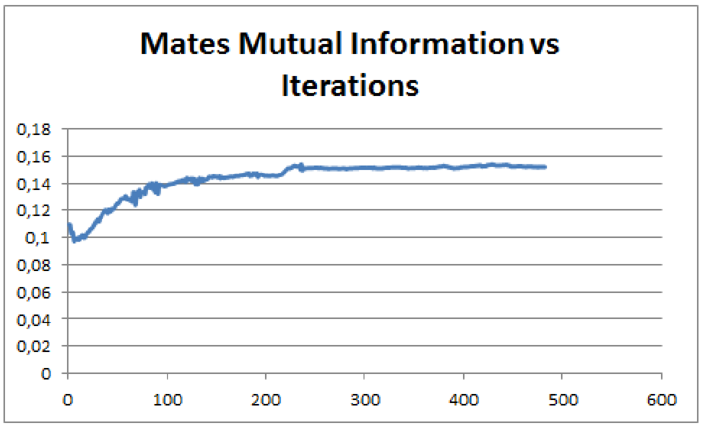

(d) Comparison between (b) and (c)

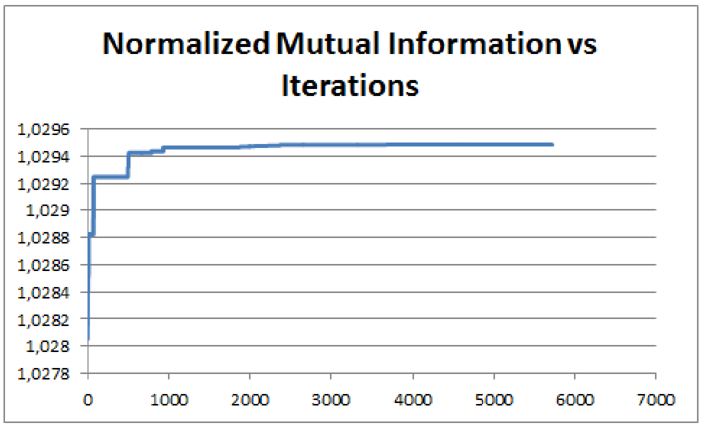
Figure 20: Successful MRT1-to-Angiogram Registration



(a) Extended Diagram

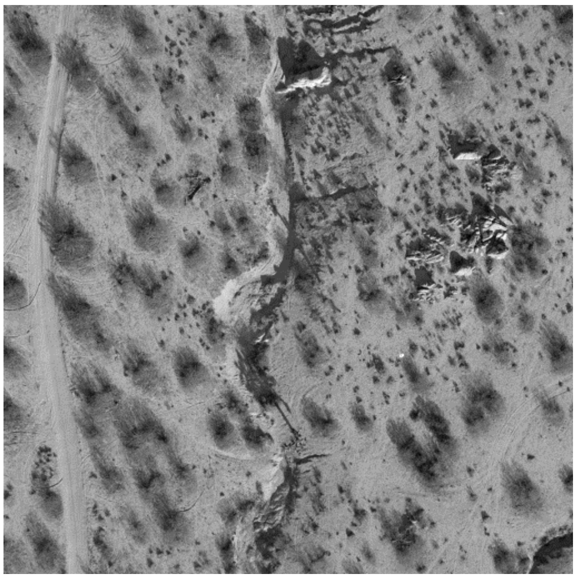


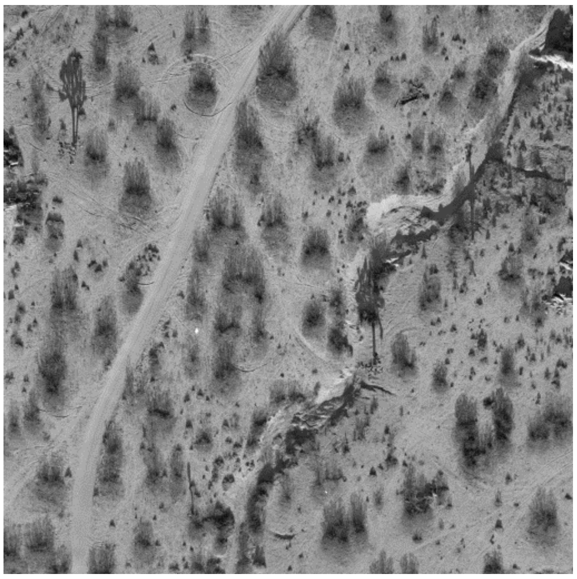
(b) ITK method of Mates Mutual Information

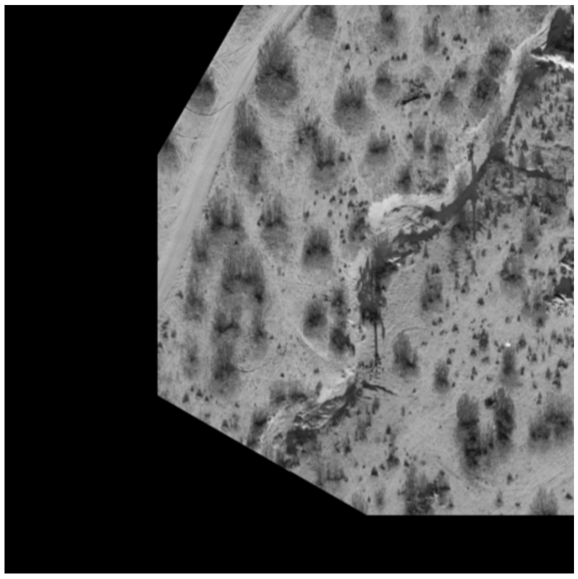


(c) ITK method of Normalized Mutual Information

Figure 21: Diagrams of MRT1-to-Angiogram Registration


(a) Floating Image


(b) Reference Image


(c) Successful Registration

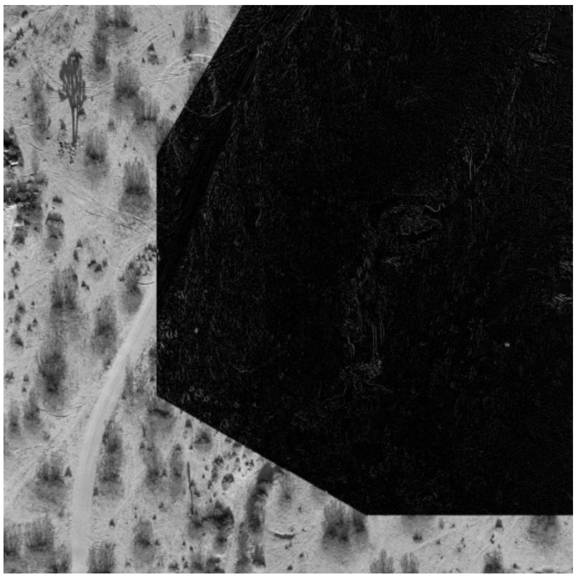
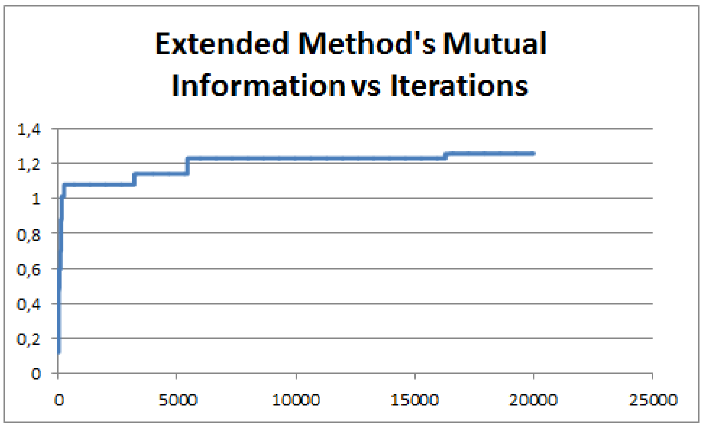
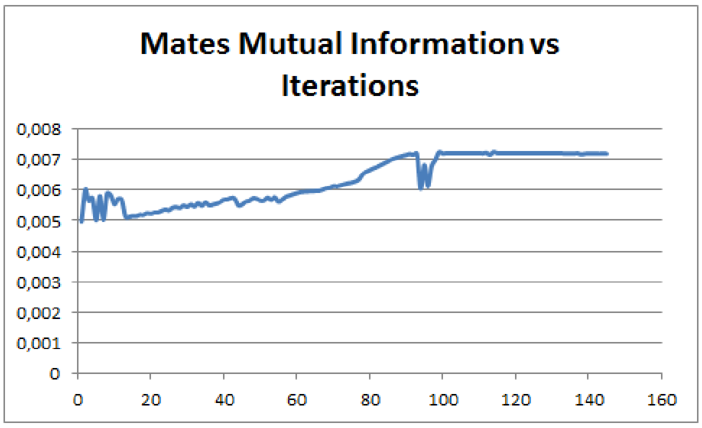

(d) Comparison between (b) and (c)

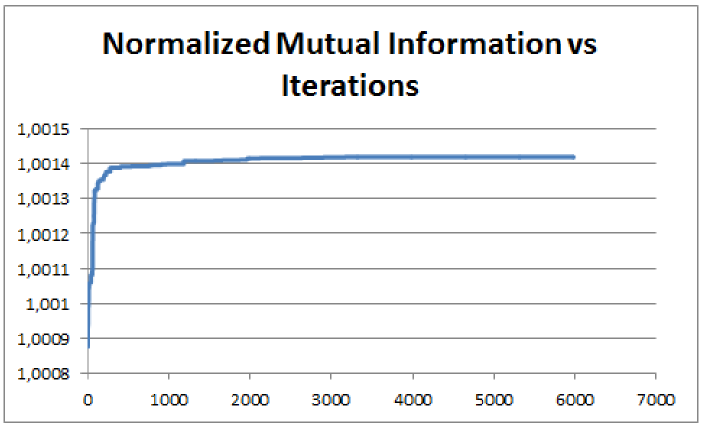
Figure 22: Successful Landscape Registration



(a) Extended Diagram

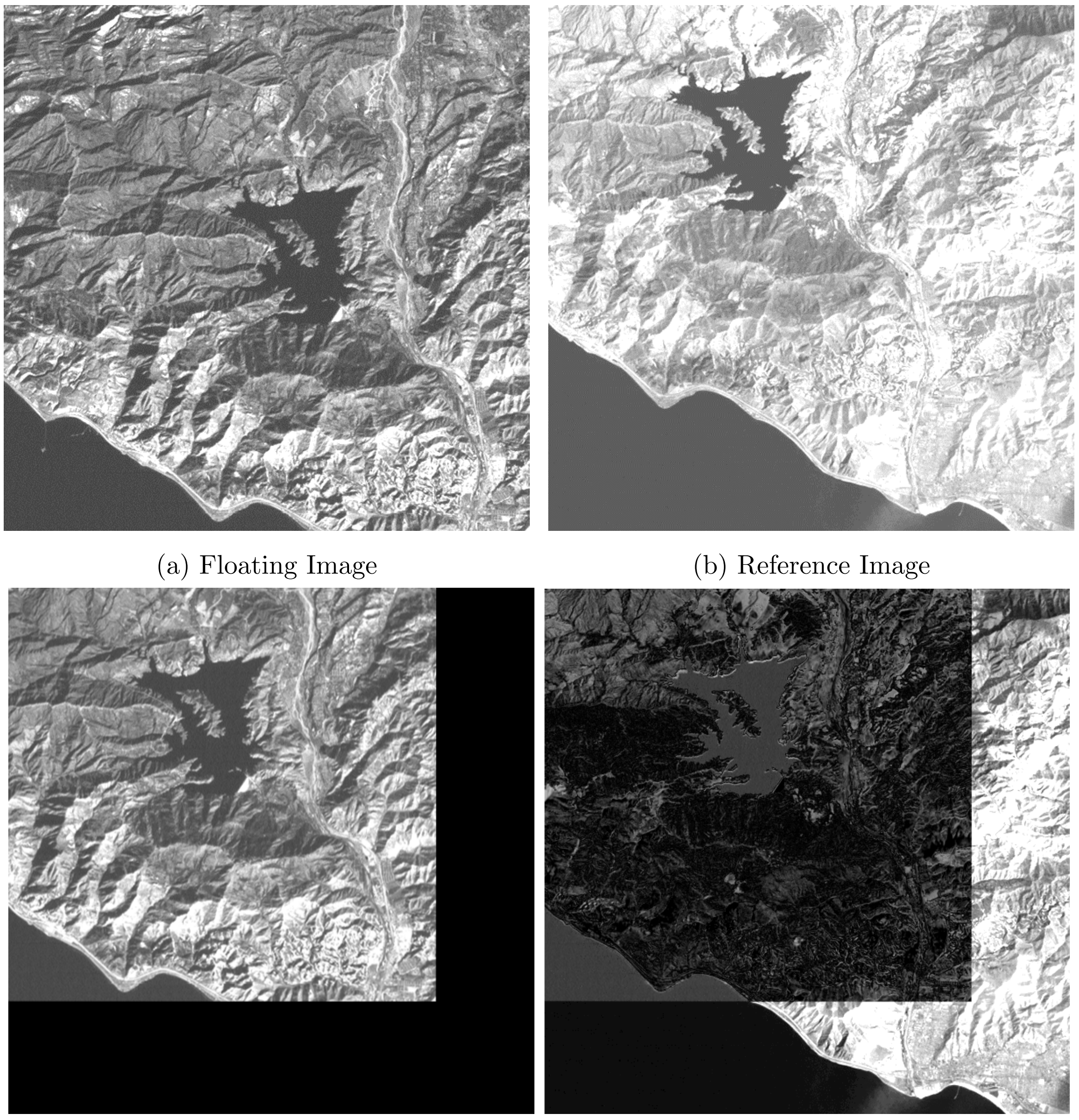


(b) ITK method of Mates Mutual Information



(c) ITK method of Normalized Mutual Information

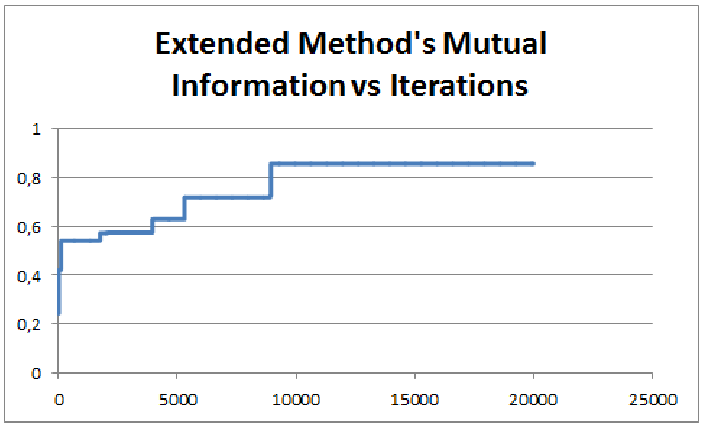
Figure 23: Diagrams of Landscape Registration



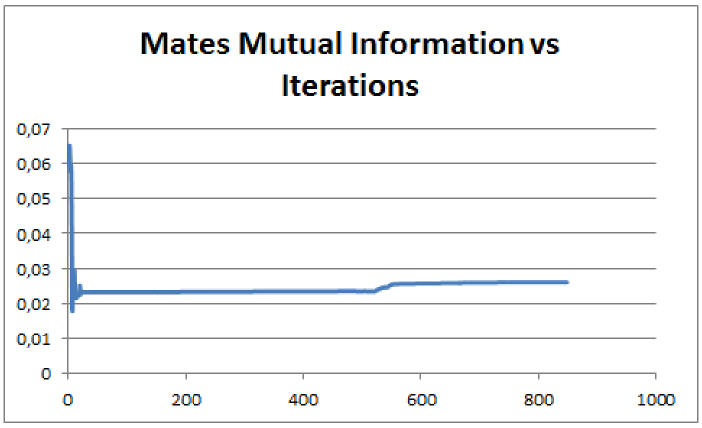
(a) Floating Image (b) Reference Image

(c) Successful Registration (d) Comparison between (b) and (c)

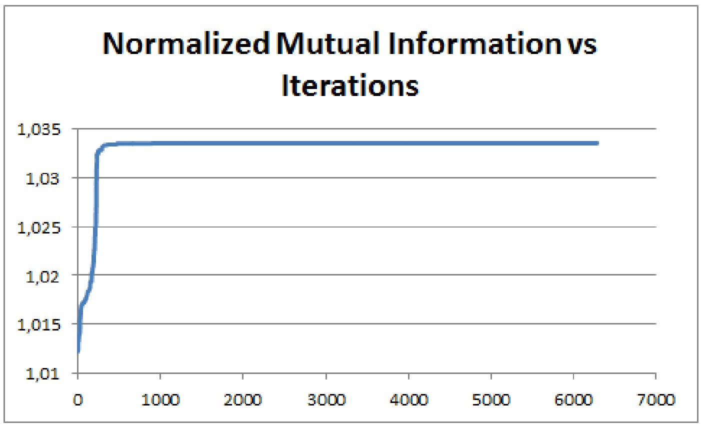
Figure 24: Successful Multi-modal Registration



(a) Extended Diagram

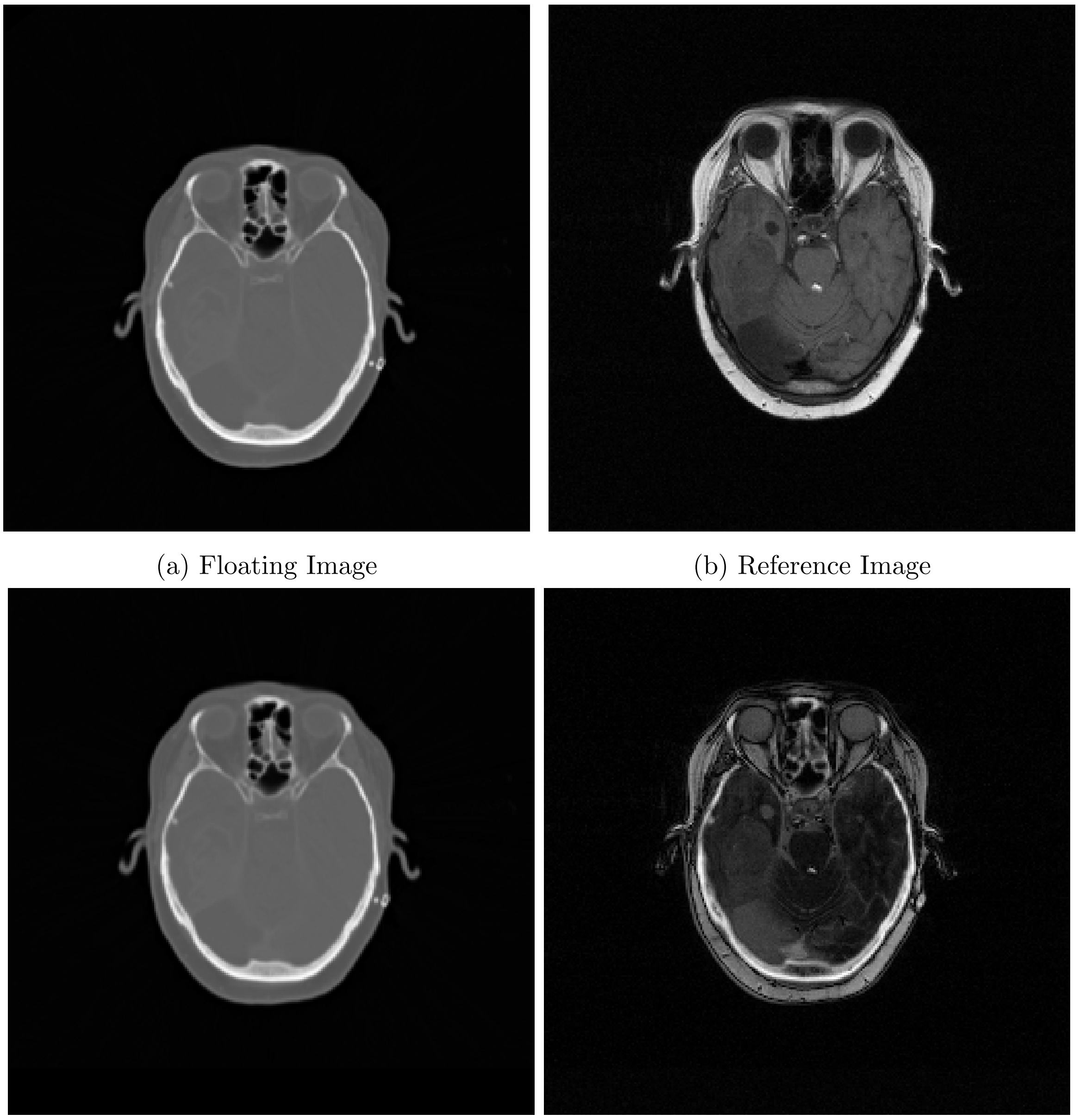


(b) ITK method of Mates Mutual Information



(c) ITK method of Normalized Mutual Information

Figure 25: Diagrams of Multi-Modal Registration

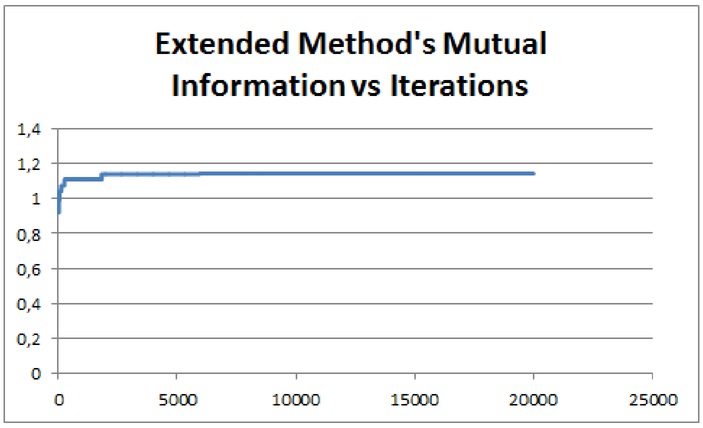

(a) Floating Image

(b) Reference Image

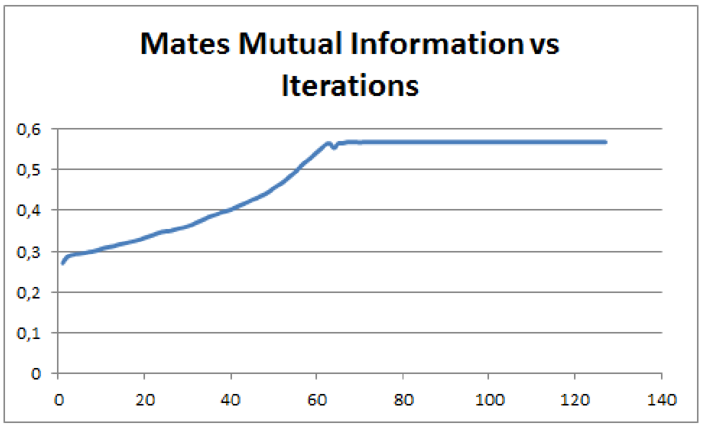
(c) Successful Registration

(d) Comparison between (b) and (c)

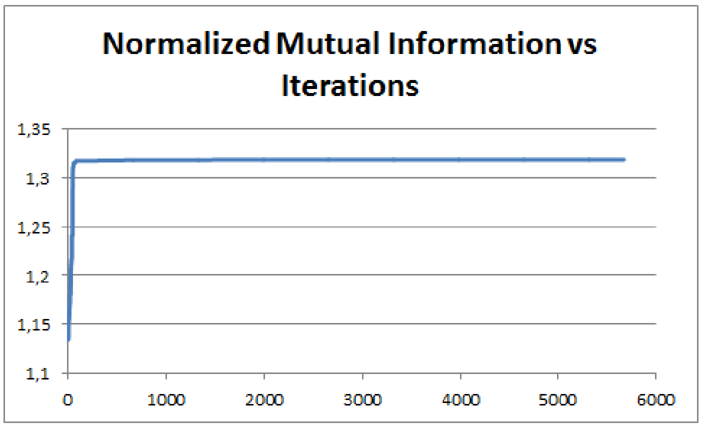
Figure 26: Successful CT-to-MRT1 Registration



(a) Extended Diagram

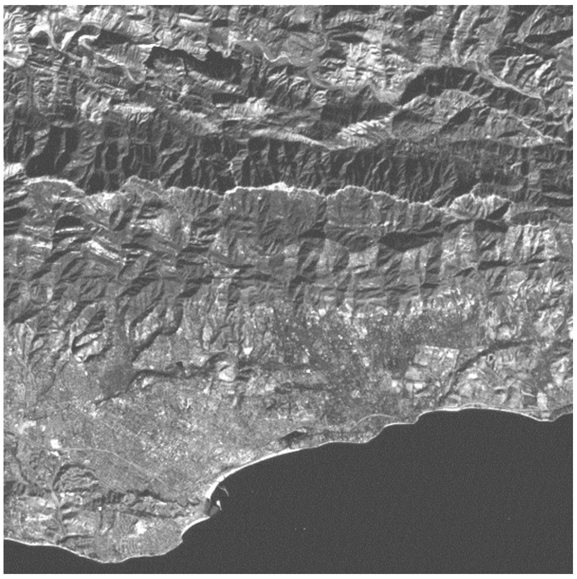


(b) ITK method of Mates Mutual Information

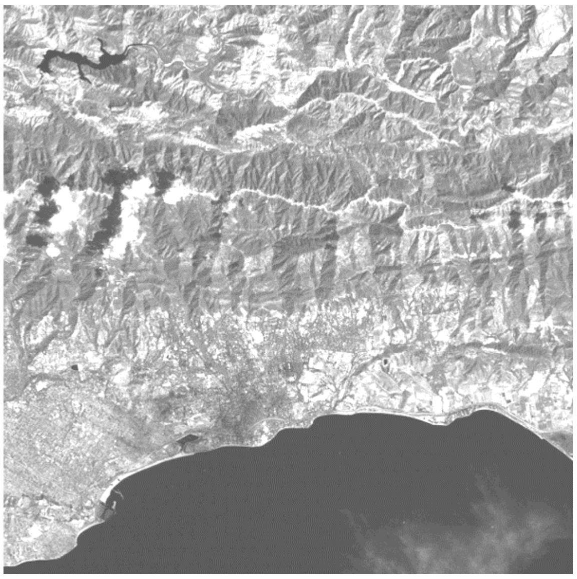


(c) ITK method of Normalized Mutual Information

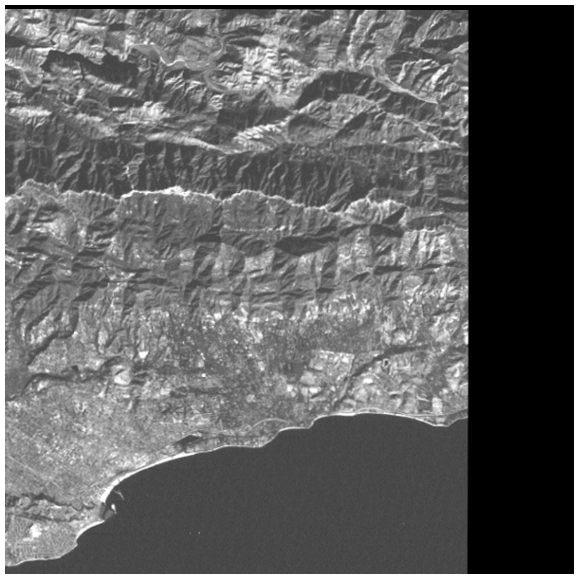
Figure 27: Diagrams of CT-to-MRT1 Registration



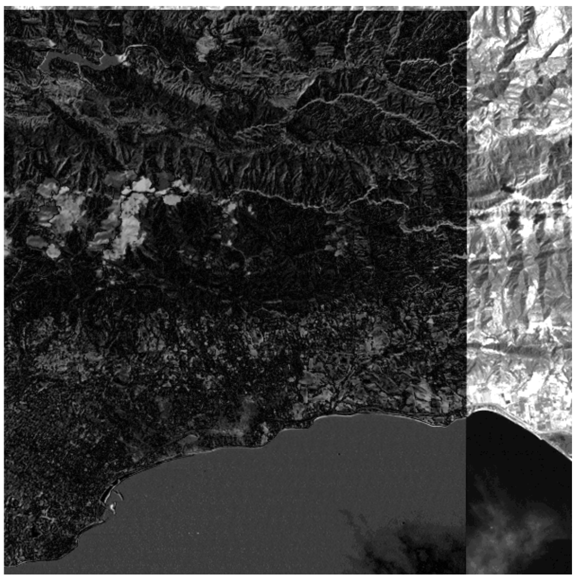
(a) Floating Image



(b) Reference Image

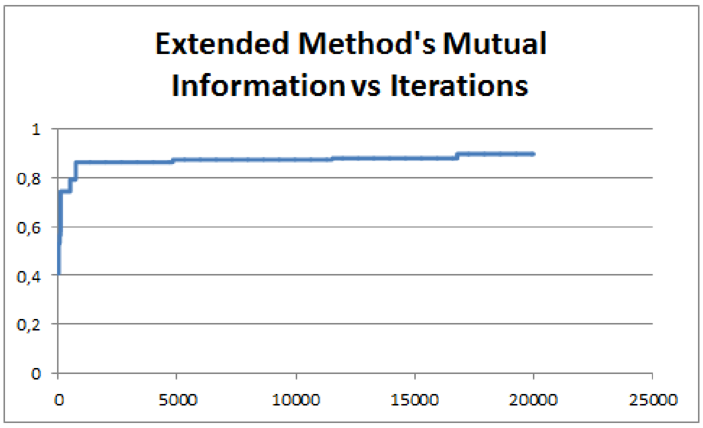


(c) Successful Registration

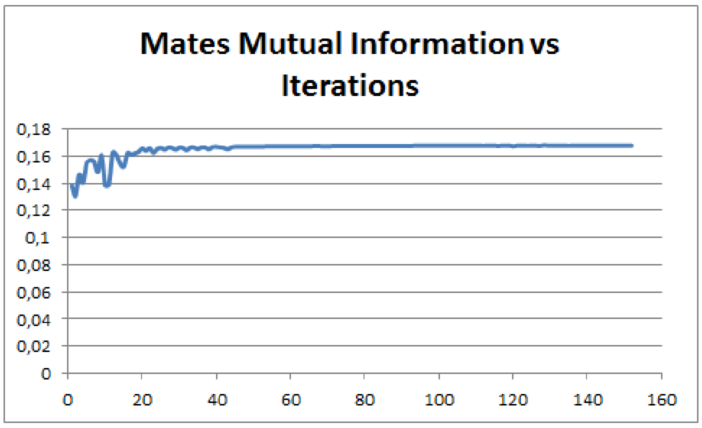


(d) Comparison between (b) and (c)

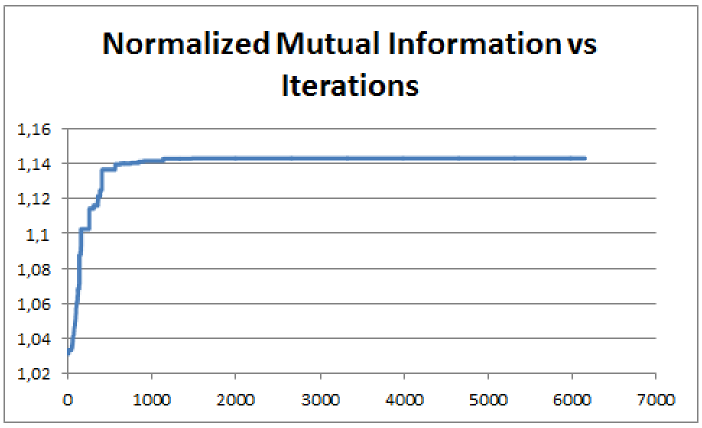
Figure 28: New Successful Multi-modal Registration



(a) Extended Diagram

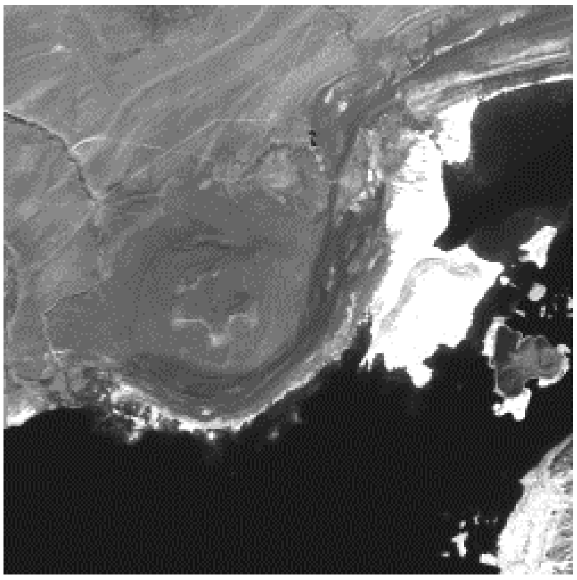


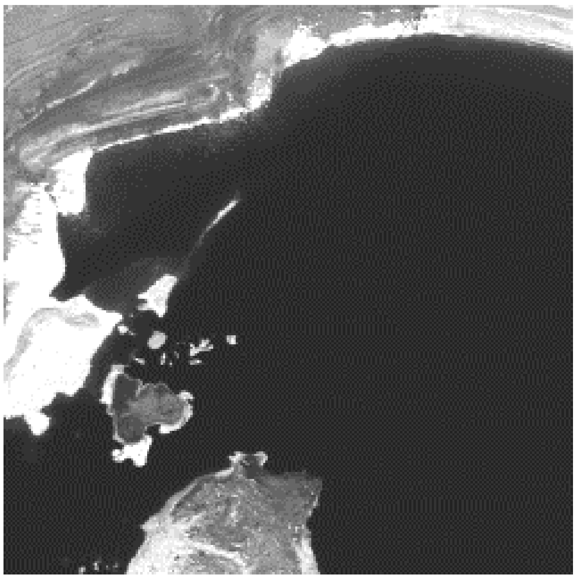
(b) ITK method of Mates Mutual Information

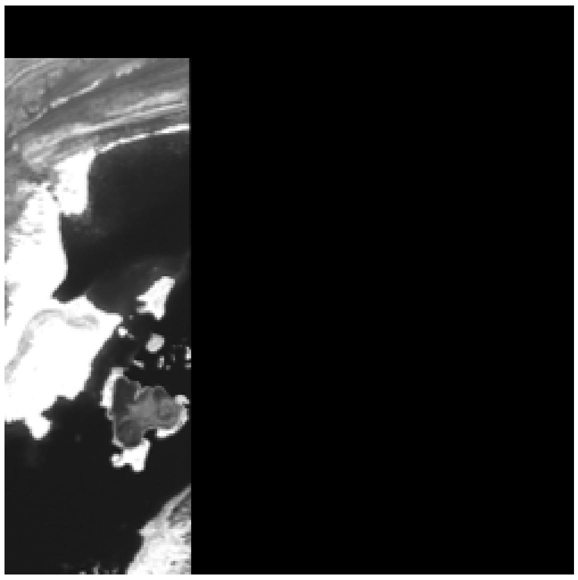


(c) ITK method of Normalized Mutual Information

Figure 29: Diagrams of Multi-Modal Registration


(a) Floating Image


(b) Reference Image


(c) Successful Registration

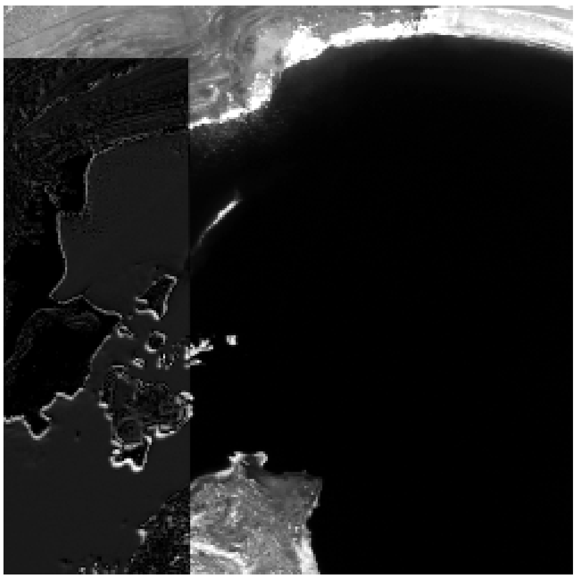
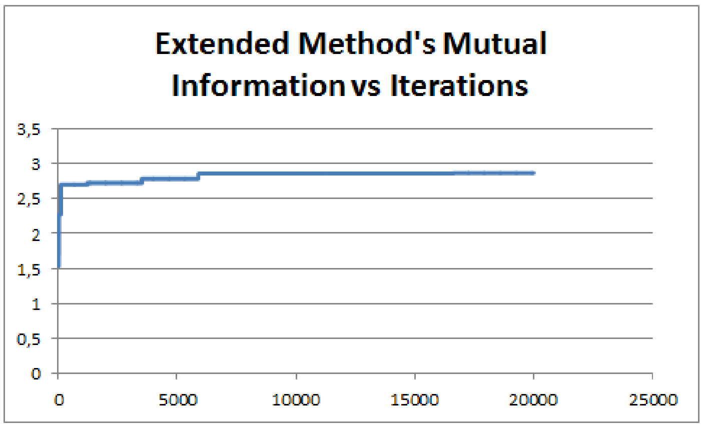
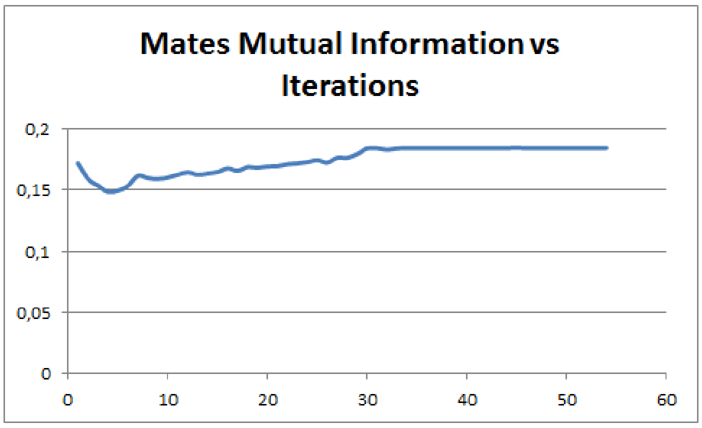

(d) Comparison between (b) and (c)

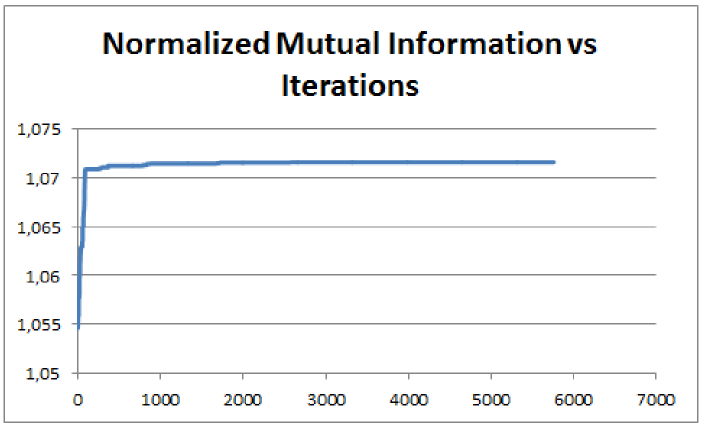
Figure 30: Successful Landscape Registration



(a) Extended Diagram

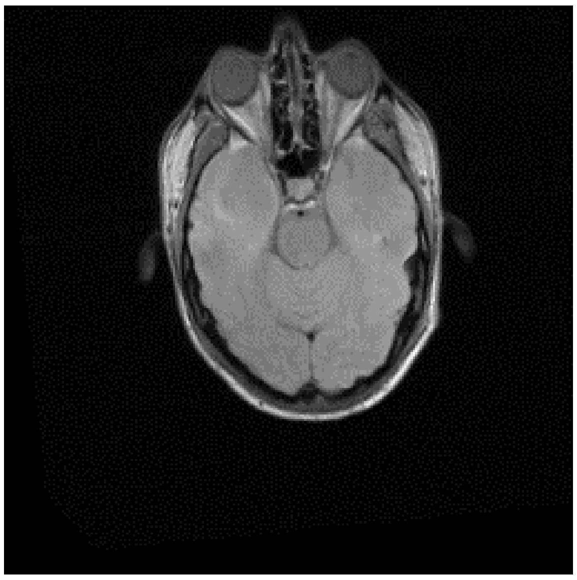


(b) ITK method of Mates Mutual Information

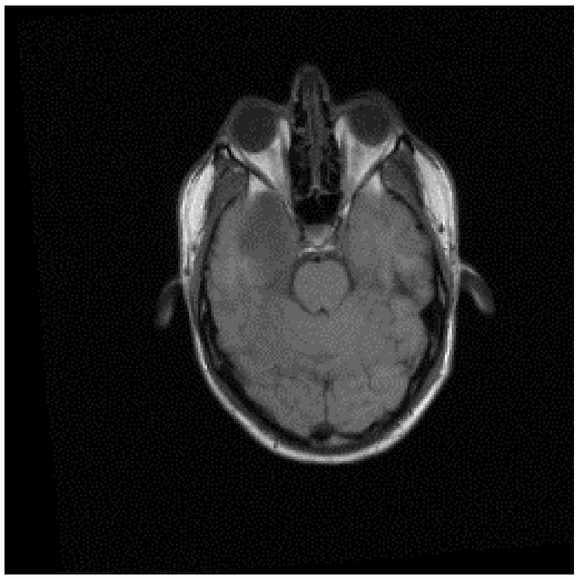


(c) ITK method of Normalized Mutual Information

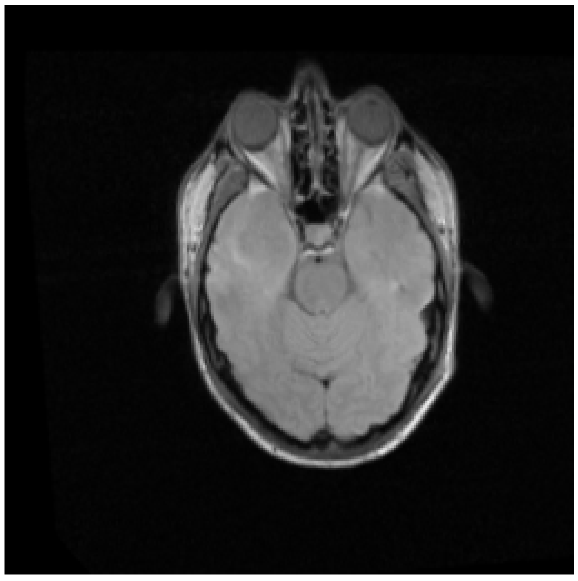
Figure 31: Diagrams of Landscape Registration



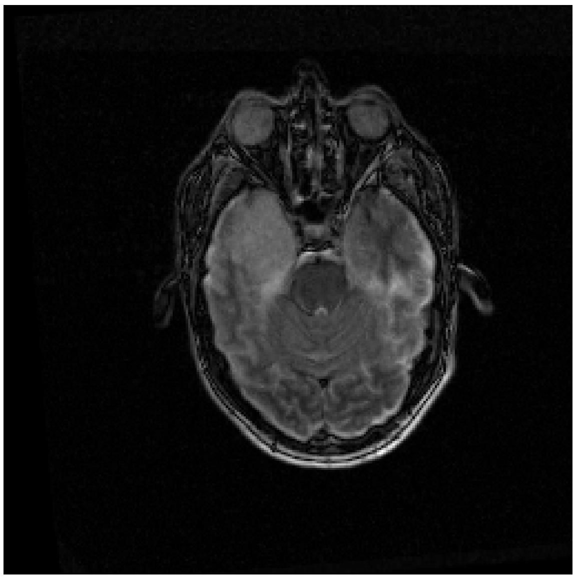
(a) Floating Image



(b) Reference Image

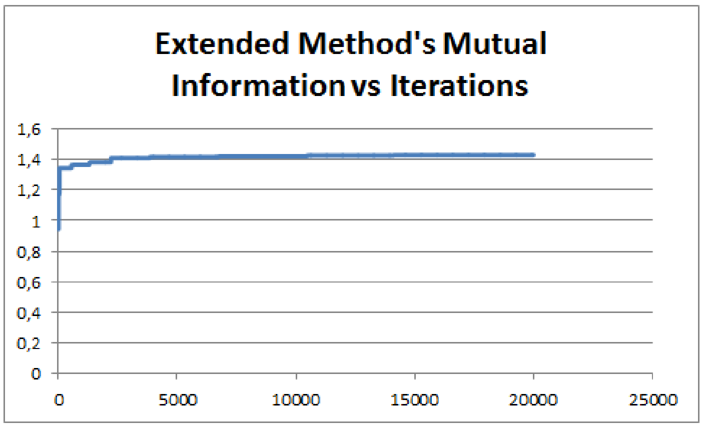


(c) Successful Registered Original Image

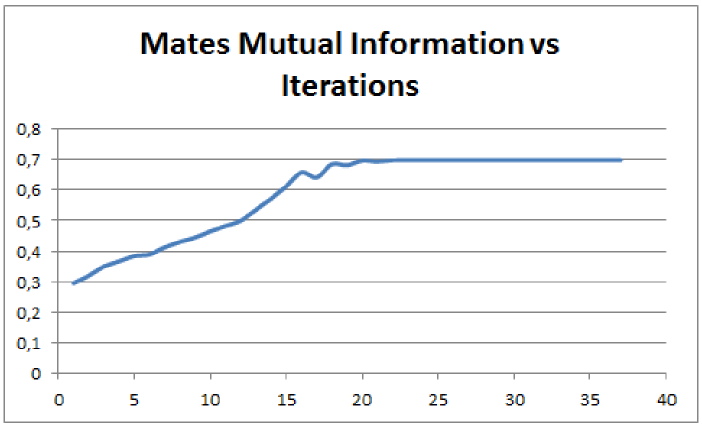


(d) Comparison between (b) and (c)

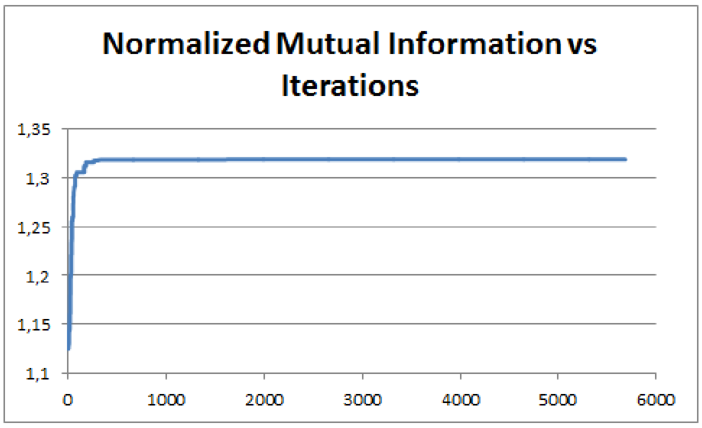
Figure 32: Successful MRPD-to-MRT1 Registration



(a) Extended Diagram

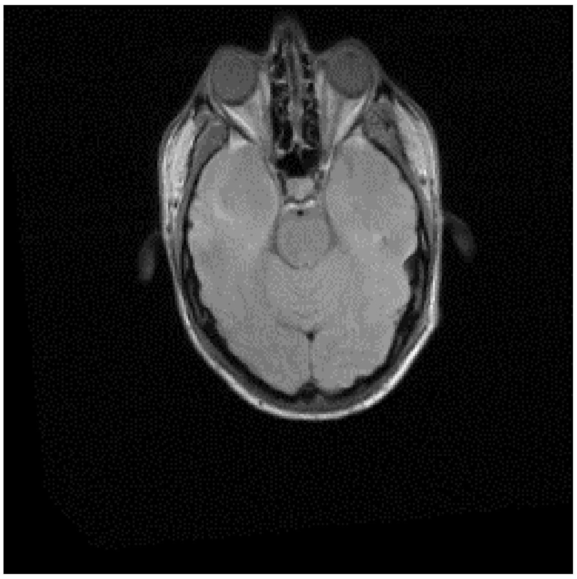


(b) ITK method of Mates Mutual Information

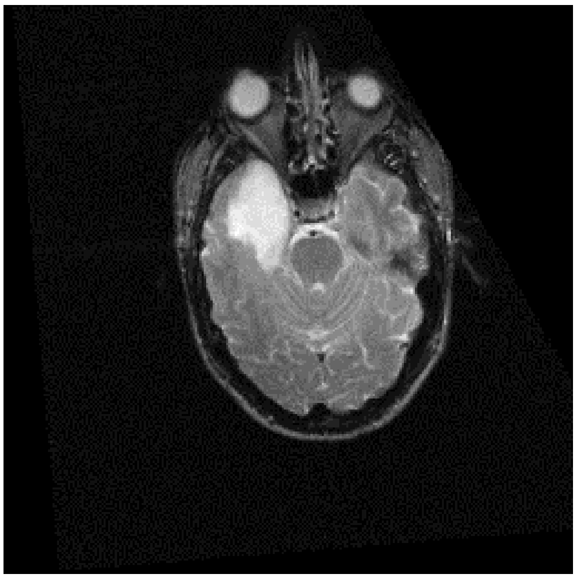


(c) ITK method of Normalized Mutual Information

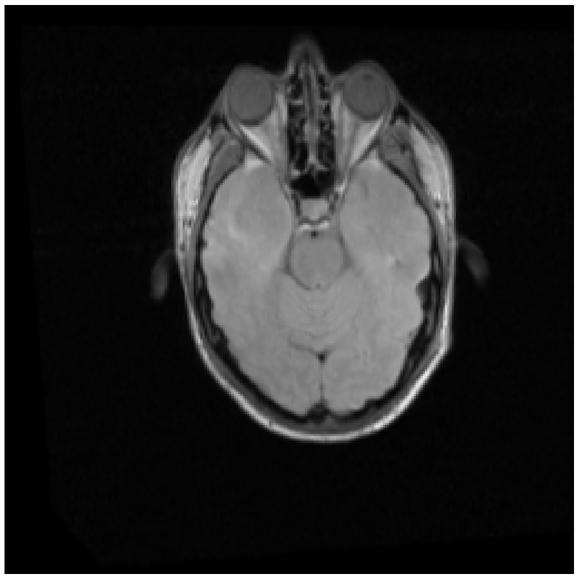
Figure 33: Diagrams of MRPD-to-MRT1 Registration



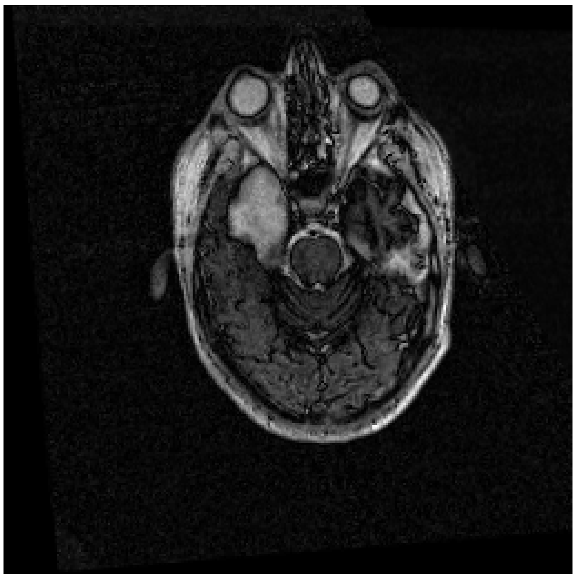
(a) Floating Image



(b) Reference Image

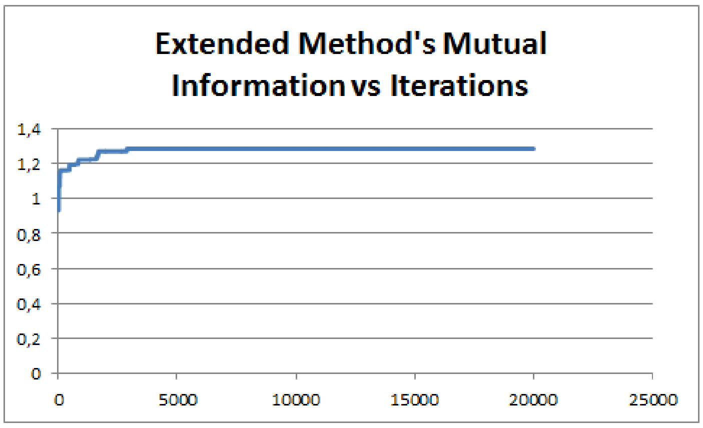


(c) Successful Registration

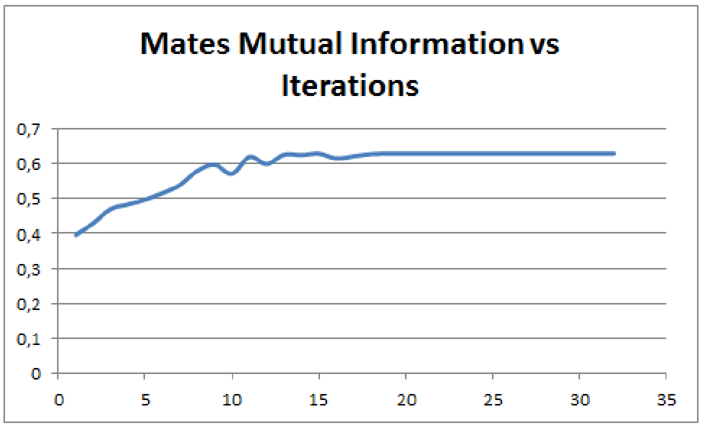


(d) Comparison between (b) and (c)

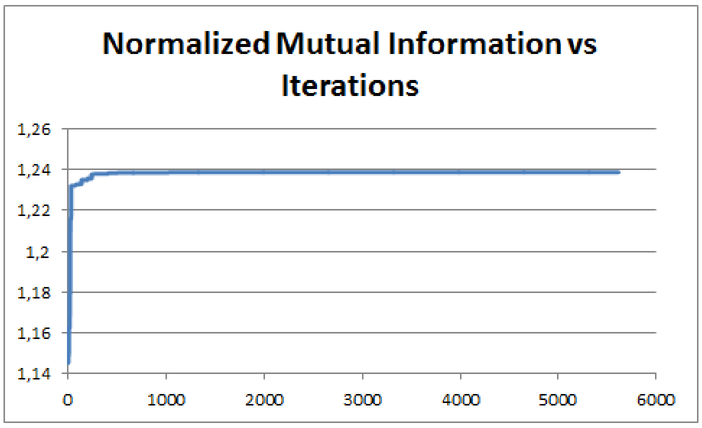
Figure 34: Successful MRPD-to-MRT2 Registration



(a) Extended Diagram

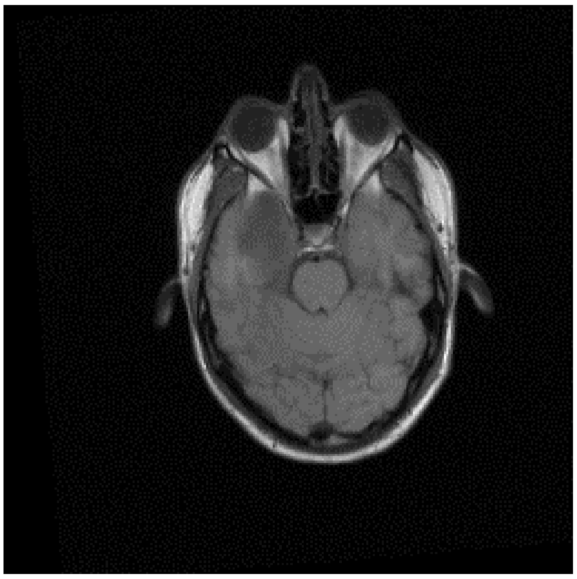


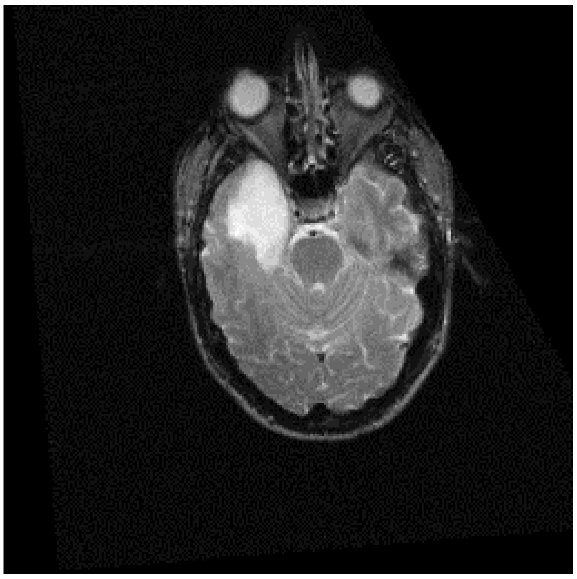
(b) ITK method of Mates Mutual Information

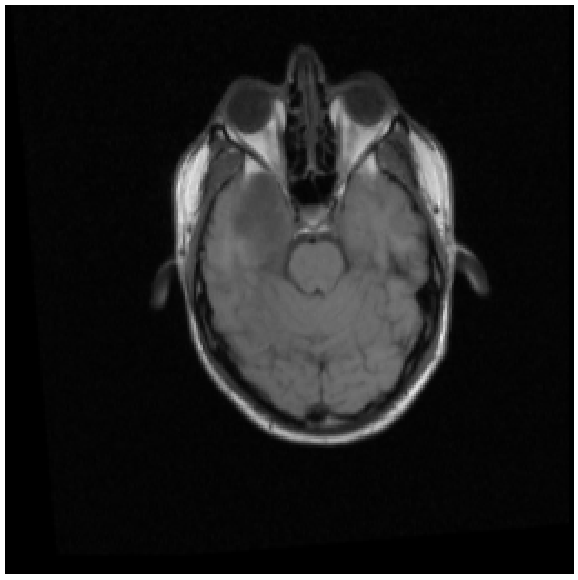


(c) ITK method of Normalized Mutual Information

Figure 35: Diagrams of MRPD-to-MRT2 Registration


(a) Floating Image


(b) Reference Image


(c) Successful Registration

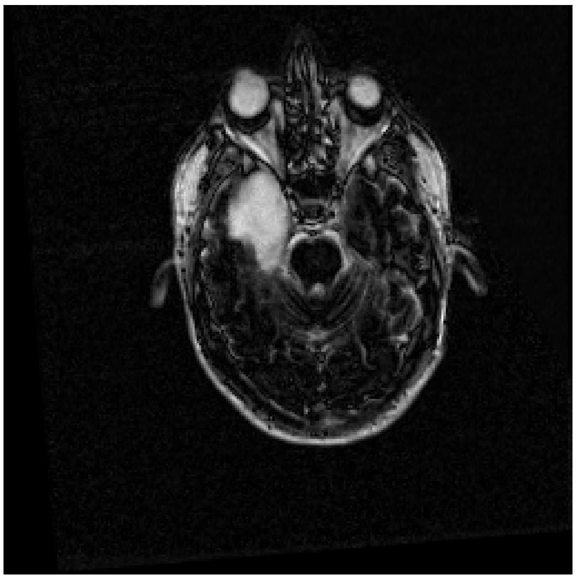
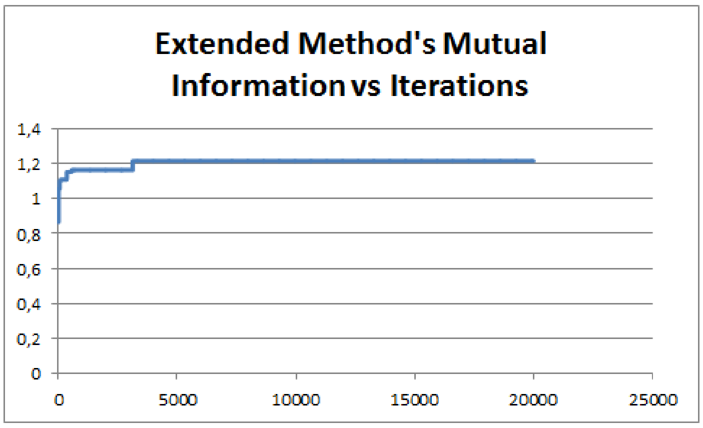
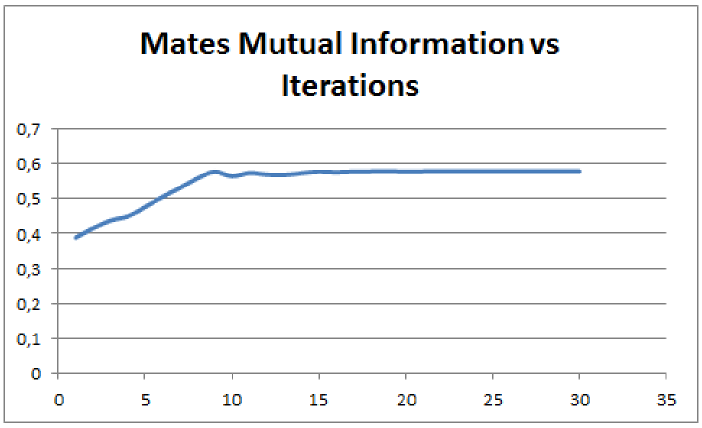

(d) Comparison between (b) and (c)

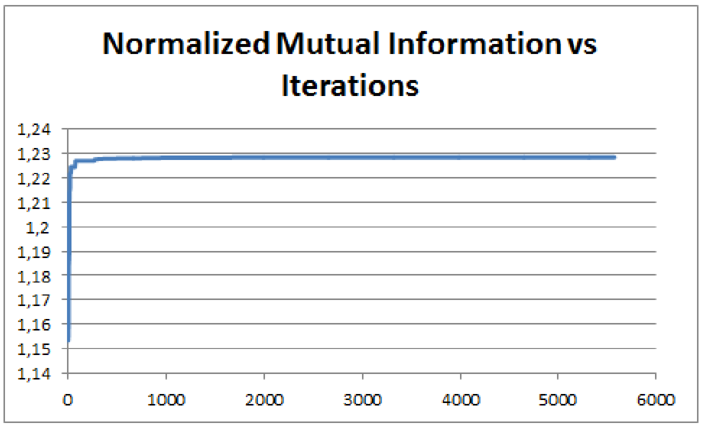
Figure 36: Successful MRT1-to-MRT2 Registration



(a) Extended Diagram

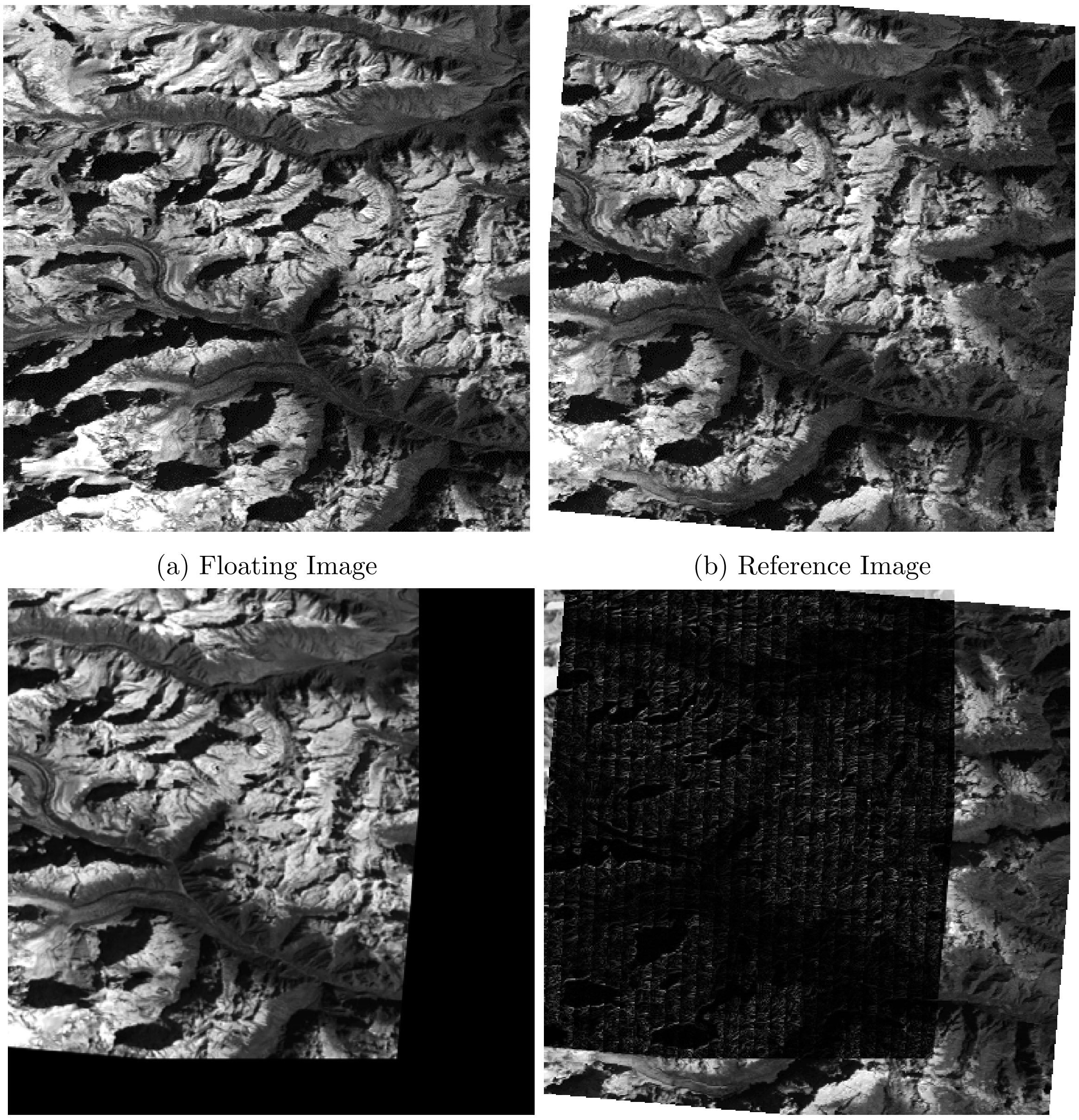


(b) ITK method of Mates Mutual Information



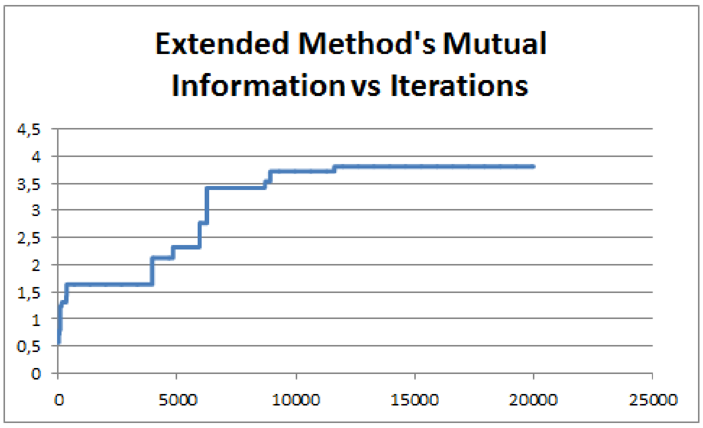
(c) ITK method of Normalized Mutual Information

Figure 37: Diagrams of MRT1-to-MRT2 Registration

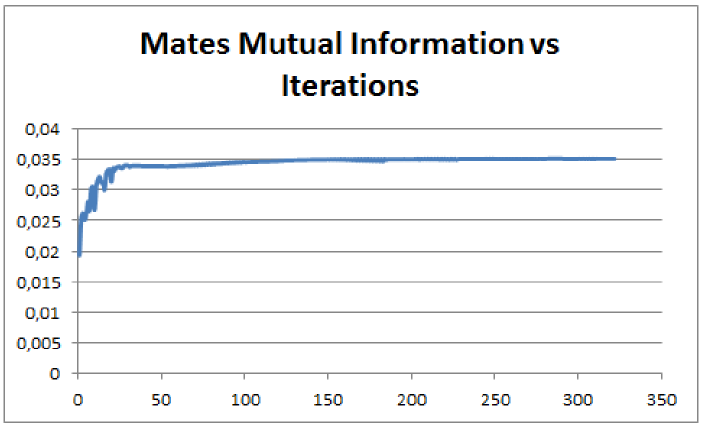

(a) Floating Image (b) Reference Image

(c) Successful Registration (d) Comparison between (b) and (c)

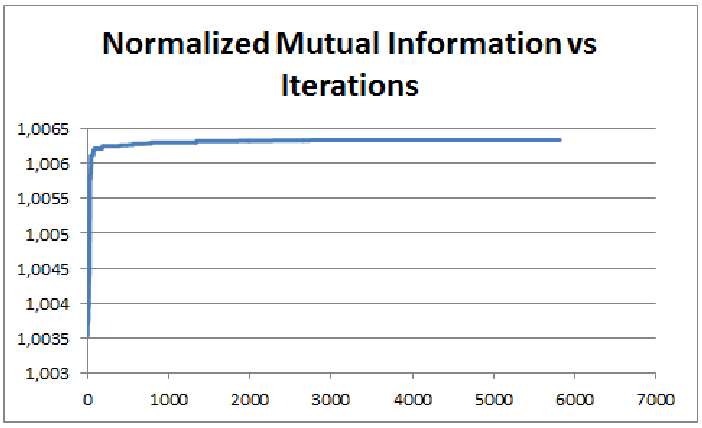
Figure 38: Successful Registration of Mountainous Area



(a) Extended Diagram

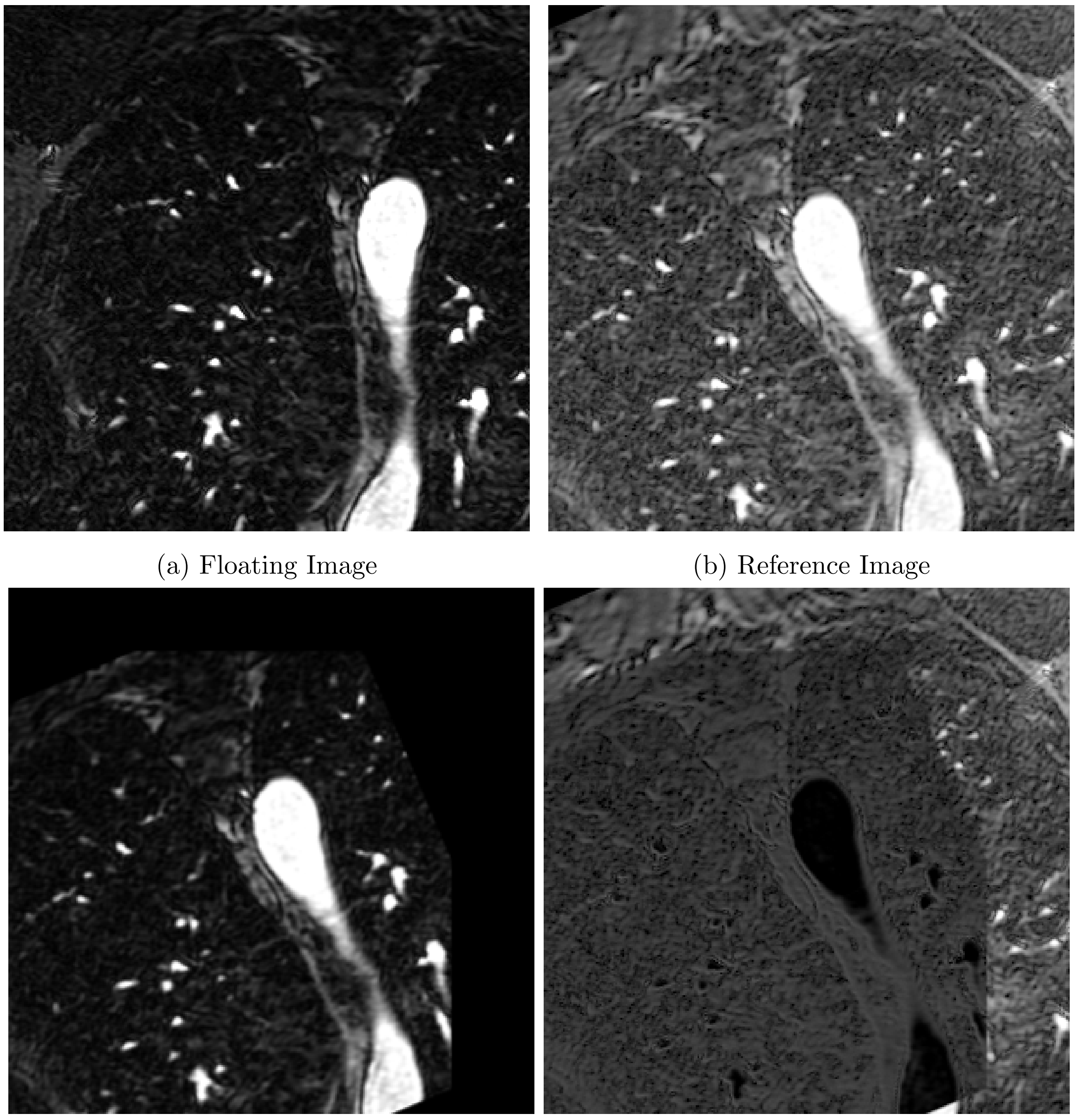


(b) ITK method of Mates Mutual Information



(c) ITK method of Normalized Mutual Information

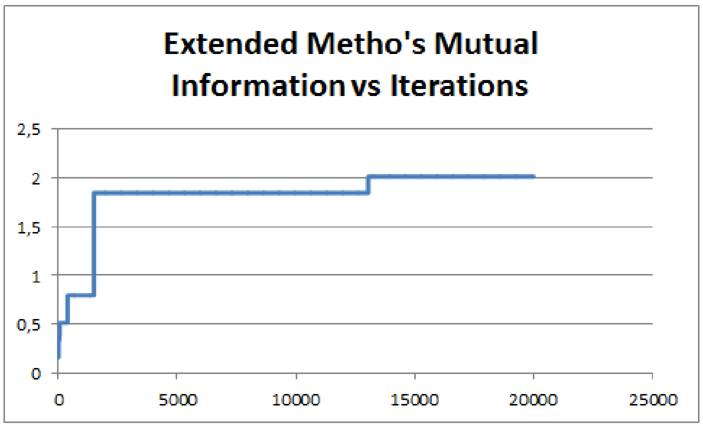
Figure 39: Diagrams of Mountainous Area Registration



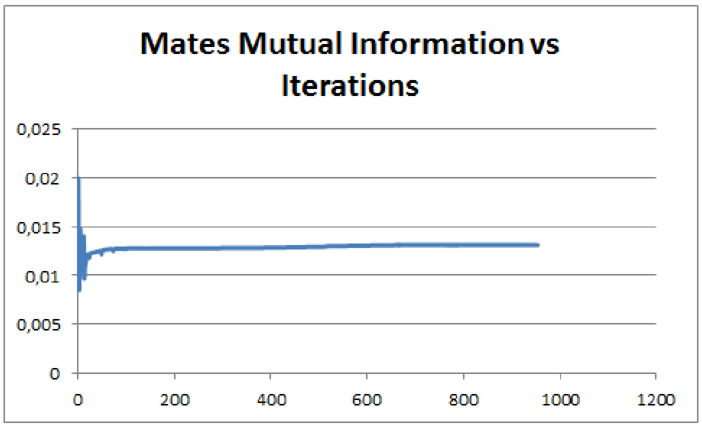
(a) Floating Image (b) Reference Image

(c) Successful Registration (d) Comparison between (b) and (c)

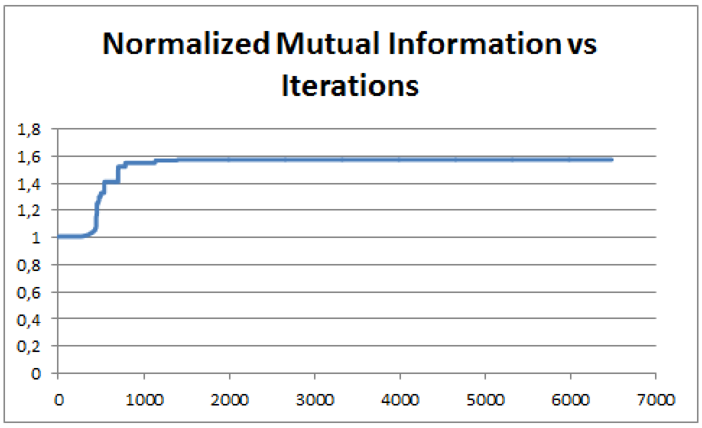
Figure 40: Successful Registration of Lung Area



(a) Extended Diagram



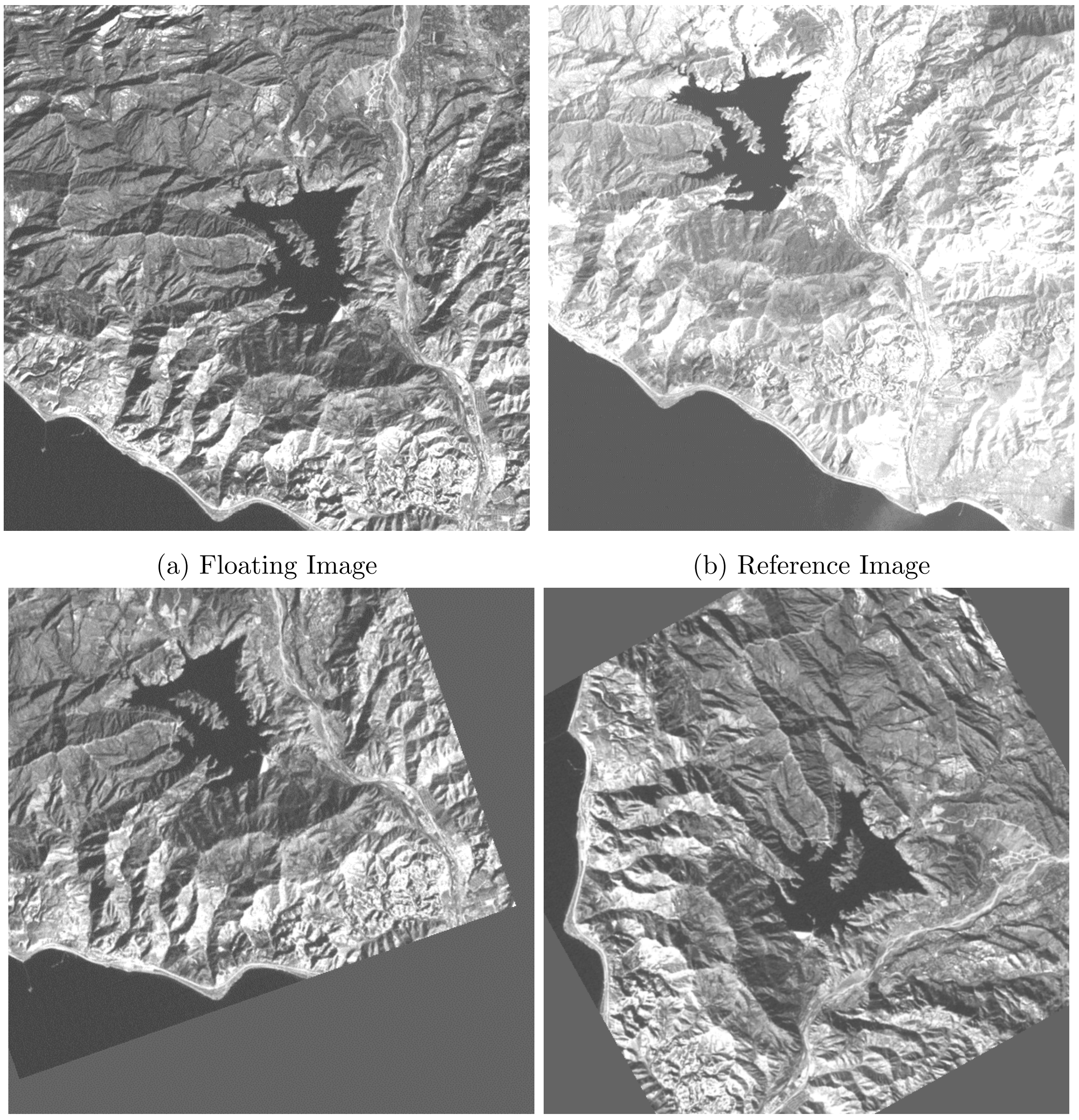
(b) ITK method of Mates Mutual Information



(c) ITK method of Normalized Mutual Information

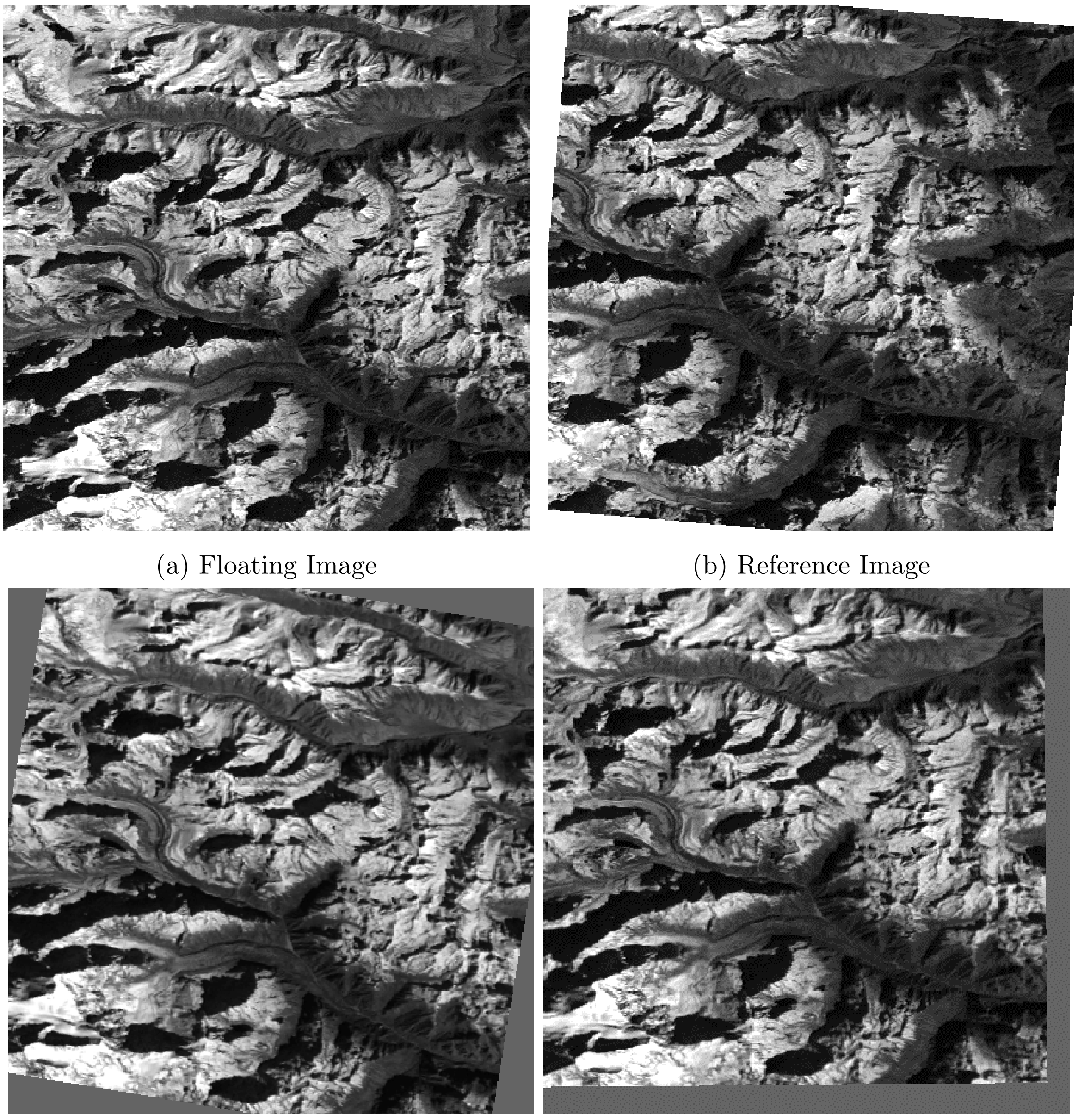
Figure 41: Diagrams of MRT1-to-MRT2 Registration

Now the failed registrations that were computed by ITK are presented


(a) Floating Image (b) Reference Image

(c) Failed use of NMI (d) Failed use of Mates MI

Figure 42: Failed Registration of Landscape area


(a) Floating Image (b) Reference Image

(c) Failed use of NMI (d) Failed use of Mates MI

Figure 43: Failed Registration of mountainous area

In the table it is obvious that the ITK methods perform very well in the case of medical images. But when it comes to other cases such as more complex multi-modal image registration(42), at least one of the ITK methods fails to register properly the floating image to the reference. That is especially true for the ITK method that uses Mates' mutual information being optimized by means of Regular Step Gradient Descent, which fails in every case except for the medical images(32, 34, 36). From the diagrams (21b,25b,41b) that correspond to that method, it is obvious that the method using Mattes' mutual information is, just like Maes' registration method, prone to stumble upon local minima, therefore failure of registration is quite easy to happen. The second method of ITK that uses Normalized Mutual Information being optimized by One-Plus-One evolutionary algorithm deals better with the problem of local minima than the first ITK method, but even this method fails. The extended method can overcome the problem of local minima at the cost of time spent for the search and detection of the (most possibly) optimal transformation of the floating image that leads to registration. Genetic algorithms can be time consuming especially in the case of a costly (in terms of time) function that we want to optimize. In order to understand how consuming a genetic algorithm can be, a table with time it took each method to register images in a few representative experiments example is presented.

Experiments\Method	Mattes	NMI	Extended
physio	162	2264	63361
MRT1→Angiogram	81	3808	103120
b040→b042	19	6724	155847
casitas84→casitas86	59	7347	196511
CT→MRT1	11	1779	43651
gibralt84→gibralt86	14	8598	223714
mono1→mono3	4	1566	28183
MRPD→MRT1	3	1501	44379
MRPD→MRT2	2	1546	48011
MRT1→MRT2	2	1809	44596
mtn1→mtn3	20	3985	109978

Table 2: Table with durations of experiments using ITK and the extended method

As we see from the table, the ITK method that uses the Mattes' Mutual Information optimized using Regular Step Gradient descent, seems very fast but it succeeds only in the medical imaging. The reason of the quick termination is the detection of a minimum (local or global). The ITK method that uses Normalized Mutual Information being optimized by means of One Plus One Evolutionary algorithm, tends to perform much better than the other ITK method (despite being significantly slower), even in a few cases non-medical images, but even that method fails. Finally, the extended method tends to be extremely time consuming even in the case of small images. The reason for that is the great cost of the evaluation of the mutual information. Yet, despite its great cost in time, the extended method manages to register successfully every pair of images in the experiments above. In order to do so, a high probability of crossover and mutation was used, due to the fact that the function of mutual information has many local minima (also the elitist variant and the proportionate selection method were used). That means that given time, the extended method will find a solution (albeit not the best one but one close to it. After all, genetic algorithms don't guarantee that the global optimum will be found). Therefore, it can be used in applications where robustness is required and time is not an issue, such as medical image registration.

Conclusions

In this thesis, an extended method of image registration was presented. The solution of the local-optima problem that was used is genetic algorithms. The diagrams and the figures show a considerate improvement in cases where not only Maes' method, but also ITK Methods don't reigster successfully the images. As it is stated above, genetic algorithms can be time consuming as it is shown in the table. Despite that, the robustness of the method can make it appealing for those who prefer quality over performance. Of course, there is room for improvement of the method. The next step is to decrease the duration of the optimal transformation search, which can be done either by parallelization of the method or using other methods of reducing the cost of function evaluation such as fitness approximation.

Appendices

Code of Maes' Method

In this section, the code is presented along with comments in order to understand its function.

```
#include<stdio.h>
#include<stdlib.h>

#include<time.h>
#include "nrutil.h"
#include "nr.h"
//#include<omp.h>
#include<math.h>


/*size of Floating and Reference image*/
#define F_M 512
#define F_N 512
#define R_M 512
#define R_N 512

int step=1;

#define NUM_THREADS 4
#define NUM_START 1
#define NUM_END 10

#define PI 3.141592653589793

unsigned char **F,**R;
//float sum;


struct point{
    double x,y;
};



void readImages();
float roundf(float x);
float NN_approx(float arg[]);
float bilinear_approx(float arg[]);
float pvi_approx(float arg[]);
float fun2(float arg[]);
float fun3(float arg[]);
float fun4(float arg[]);



int main(){
        float p[2],**xi,tol=0.000000000001,fret=0,p2[3],x[3],temp[2500][3],res,l;
        int n=-1,iter=100,i,j,k,n2=3,n1=2;
        double hx=50.0/400.0, hy=50.0/400.0, hangle=2*PI/400,init;
        time_t now,now2;

        double seconds=0;
        FILE* f;

        /*Read the images from text files */
        readImages();

        //omp_set_num_threads(NUM_THREADS);



        /*Initialization of direction vectors*/
        xi=(float**)malloc(n2*sizeof(float*));

        for(i=0;i<n2;i++){
            xi[i]=(float*)malloc(n2*sizeof(float));

        }


        /*Set values of direction vectors*/
        for(i=0;i<n2;i++){
            for(j=0;j<n2;j++){
                xi[i][j]=0;
            }
        }
        xi[0][1]=xi[1][0]=xi[2][2]=1;


        /*Initial transformation matrix*/
        p2[0]=0;
        p2[1]=0;
        p2[2]=0;

        printf("Initial Function Value is %f\n",pvi_approx(p2));

        powell(p2,xi,n2,tol,&iter,&fret,pvi_approx);
        res=fret;

        x[0]=p2[0];
        x[1]=p2[1];
        x[2]=p2[2];
        printf("%lf\n",hx);
        f=fopen("myfile.txt","w");
        for(init=-50.0;init<=50.0; init+=hx){
            printf("%lf\n",init);
            p2[2]=init;
            if(init==0.0){
                printf("i am zero\n");
            }
            p2[0]=0;
            p2[1]=0;
            fprintf(f,"%f %f %f\n",NN_approx(p2),bilinear_approx(p2),pvi_approx(p2));

        }

        fclose(f);

        printf("p2 is (%17.15f,%17.15f,%17.15f)\n and fret is %17.15f after %d iterations\n↩
            ",x[0],x[1],x[2],res,iter);
        system("PAUSE");
        return 0;
 }

/*Return rounded integer value of a float number*/
float roundf(float x){
    return floor(x + 0.5);
}







/*Rosenbrock's Test function that shows the problem of Powell's method*/
float fun2(float arg[]){
    return 100*(arg[1]-arg[0]*arg[0])*(arg[1]-arg[0]*arg[0])+(arg[0]-1)*(arg[0]-1);
}


/*Test function that shows the problem of Powell's method*/
float fun3(float arg[]){
    return arg[0]*arg[1]*exp(-arg[0]*arg[0]-arg[1]*arg[1]);
}

/*Ackley's Test function that shows the problem of Powell's method*/
float fun4(float arg[]){
    return -20*exp(-0.2*sqrt(0.5*(arg[0]*arg[0]+arg[1]*arg[1])))-exp(0.5*(cos(2*PI*arg[0])+↩
        cos(2*PI*arg[1])))+20+exp(1);
}



/*Function readImages() reads the contents of images that are
stored in txt files myFile3.txt (which is the Floating image)
and myFile4.txt (which is the  reference image) and stores
them in the dynamic matrices F and R respectively*/

void readImages(){
    unsigned int  i,j,x=1,y=1;
    FILE *f1,*f2,*f3;


    //Floating image
    F=(unsigned char**)malloc(F_M*sizeof(unsigned char*));

    //Reference Image
    R=(unsigned char**)malloc(R_M*sizeof(unsigned char*));



    //Memory Allocation
    for(i=0;i<F_M;i++){
        F[i]=(unsigned char*)malloc(F_N*sizeof(unsigned char));
        R[i]=(unsigned char*)malloc(R_N*sizeof(unsigned char));
    }


    f1=fopen("myFile3.txt","r");
    f2=fopen("myFile4.txt","r");
    if(f1==NULL || f2==NULL ){
        printf("problem1\n");
        return;
    }
    i=j=0;

    //Read File for Floating Image
    while(!feof(f1)){
        fscanf(f1,"%uc",&F[i][j]);
        j++;
        if(j==F_N){
            j=0;
            i++;
        }
    }
    fclose(f1);

    i=j=0;


    //Read File for Reference Image
    while(!feof(f2)){
        fscanf(f2,"%uc",&R[i][j]);
        j++;
        if(j==R_N){
            j=0;
            i++;
        }
    }
    fclose(f2);
}


/*Calculation of Mutual Information using Nearest Neighbour
approximation. It is a crude way to calculate Mutual Information,
prone to give a false image of the similarity of the Images

The variables are:
    input:
        arg[]: the parameters of the transformation operator
    output:
        Mutual information
    res:    the joint histogram
    Pr[i]:  probability of intensity i in Reference image
    Pf[i]:  probability of intensity i in Floating image
    Hf, Hr: marginal entropy of Floating and Reference image respectively
    Hrf:    joint entropy
    cF, cR: centers of Floating and Reference image respectively
    TFpoint: transformed coordinates of point in Floating image
*/

float NN_approx(float arg[]){
    unsigned int  i,j,x=1,y=1,xb,yb;


    float a,e,f,Pr[256],Pf[256],Hf=0,Hr=0,Hrf=0;
    float res[256][256];
    float sum=0;

    struct point TFpoint,cR,cF;

    a=arg[0];//Rotation in radians

    e=arg[1];//Translation in x axis
    f=arg[2];//Translation in y axis

    //Initialize joint histogram and probability function
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            res[i][j]=0;
        }

        Pr[i]=0;
        Pf[i]=0;
    }

    //Find centres of Images
    if (R_M%2==1){
        cR.y=ceil((double)(R_M)/2);
    }
    else{
        cR.y=R_M/2;
    }


    if (R_N%2==1){
        cR.x=ceil((double)(R_N)/2);
    }
    else{
        cR.x=R_N/2;
    }


    if (F_M%2==1){
        cF.y=ceil((double)(F_M)/2);
    }
    else{
        cF.y=F_M/2;
    }


    if (F_N%2==1){
        cF.x=ceil((double)(F_N)/2);
    }
    else{
        cF.x=F_N/2;
    }



    /*Making of the joint histgram*/
    for(i=0;i<F_N;i=i+step){
        for(j=0;j<F_M;j=j+step){

            /*Transform the coordinates of the Floating image  using the
            parameters a,e,f and the centers of the images*/
            TFpoint.x=cos(a)*(i+1-cF.x)- sin(a)*(j+1-cF.y)+cR.x-1+e;
            TFpoint.y=sin(a)*(i+1-cF.x)+ cos(a)*(j+1-cF.y)+cR.y-1+f;

            /*Check only poits of Floating image that when they are transformed in space
            they are inside the space of the Reference Image*/
            if (TFpoint.x>=0 && TFpoint.x<R_N && TFpoint.y>=0 && TFpoint.y<R_M){
                y=(unsigned int)F[j][i];


                /*First case: The transformed coordinates of the Floating image'ss pixel
                are integers*/
                if(floor(TFpoint.x) ==TFpoint.x && floor(TFpoint.y) ==TFpoint.y){

                    x=R[(unsigned int)TFpoint.y][(unsigned int)TFpoint.x];

                    res[x][y]=res[x][y]+1;
                }

                /*Second case: The transformed coordinate x of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.x) ==TFpoint.x){


                    if(floor(TFpoint.y)<0){

                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];
                        res[x][y]=res[x][y]+1;
                    }
                    else if(ceil(TFpoint.y)>=R_M){

                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];
                        res[x][y]=res[x][y]+1;
                    }
                    else if(fabs(floor(TFpoint.y)-TFpoint.y)<fabs(ceil(TFpoint.y)-TFpoint.y↩
                        )){
                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];

                        res[x][y]=res[x][y]+1;

                    }
                    else{
                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];

                        res[x][y]=res[x][y]+1;

                     }
                }

                /*Third case: The transformed coordinate y of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.y) ==TFpoint.y){



                    if(floor(TFpoint.x)<0){

                        x=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];
                        res[x][y]=res[x][y]+1;
                    }
                    else  if(ceil(TFpoint.x)>=R_N){

                        x=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];
                        res[x][y]=res[x][y]+1;
                    }
                    else  if(fabs(floor(TFpoint.x)-TFpoint.x)<fabs(ceil(TFpoint.x)-TFpoint.x↩
                        )){
                        x=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];

                        res[x][y]=res[x][y]+1;
                    }
                    else{
                        x=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];

                        res[x][y]=res[x][y]+1;
                    }
                }

                /*Fourth case: The transformed coordinates of the Floating image's pixel
                aren't integers*/
                else{

                    if(floor(TFpoint.x)<0){

                        xb=1;
                    }
                    else  if(ceil(TFpoint.x)>=R_N){

                        xb=0;
                    }
                    else  if(fabs(floor(TFpoint.x)-TFpoint.x)<fabs(ceil(TFpoint.x)-TFpoint.x↩
                        )){
                        xb=0;
                    }
                    else{
                        xb=1;
                    }


                    if(floor(TFpoint.y)<0){

                        yb=1;
                    }
                    else  if(ceil(TFpoint.y)>=R_M){

                        yb=0;
                    }
                    else  if(fabs(floor(TFpoint.y)-TFpoint.y)<fabs(ceil(TFpoint.y)-TFpoint.y↩
                        )){
                        yb=0;
                    }
                    else{
                        yb=1;
                    }




                    if(xb==0 && yb==0){
                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)floor(TFpoint.x)↩
```

```
                            ];

                        res[x][y]=res[x][y]+1;
                    }
                    else if(xb==1 && yb==0){
                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                        res[x][y]=res[x][y]+1;
                    }
                    else if(xb==0 && yb==1){
                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)floor(TFpoint.x)];

                        res[x][y]=res[x][y]+1;
                    }

                    if(xb==1 && yb==1){
                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                        res[x][y]=res[x][y]+1;
                    }
                }


            }

            if(x<0 || x>255){
                printf("x is %d\n",x);
                return 0;
            }

            if(y<0 || y>255){
                printf("y is %d\n",y);
                return 0;
            }





        }
    }

    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            sum+=res[i][j];
    }

    /*Normalization of joint histogram so that the sum of its values is equal to 1*/
    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            res[i][j]=res[i][j]/sum;
    }


    /*Calculation of probabilities*/
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            Pf[i]+=res[i][j];
            Pr[j]+=res[i][j];
        }
    }

    /*Calculation of marginal and joint entropy*/
    for(i=0;i<256;i++){
        if(Pr[i]>0){
            Hr+=-Pr[i]*log(Pr[i])/log(2.0);
        }
        if(Pf[i]>0)
            Hf+=-Pf[i]*log(Pf[i])/log(2.0);
        for(j=0;j<256;j++){
            if(res[i][j]>0){
                Hrf+=-res[i][j]*log(res[i][j])/log(2.0);
            }
        }
    }

    /*Calculation of mutual information*/
    sum=Hr+Hf-Hrf;


    return -sum;

}


/*Calculation of Mutual information using bilinear interpolation approximation.
It is more accurate than Nearest Neighbour approximation, but tends ti introduce
in subpixel level new intensities that don't exist in the image. It is less
prone to give a false image of the similarity of the Images

The variables are:
    input:
        arg[]: the parameters of the transformation operator
    output:
        Mutual information
    res:    the joint histogram
    Pr[i]:  probability of intensity i in Reference image
    Pf[i]:  probability of intensity i in Floating image
    Hf, Hr: marginal entropy of Floating and Reference image respectively
    Hrf:    joint entropy
    cF, cR: centers of Floating and Reference image respectively
    TFpoint: transformed coordinates of point in Floating image
*/
float bilinear_approx(float arg[]){
    unsigned int  i,j,x=1,y=1;

    float a,e,f,Pr[256],Pf[256],Hf=0,Hr=0,Hrf=0,q,r;
    float res[256][256];
    float sum=0;

    struct point TFpoint,cR,cF;
    FILE *f1,*f2,*f3;




    a=arg[0];//Rotation in radians
    e=arg[1];//Translation on x axis
    f=arg[2];//Translation on y axis

    /*Initialize joint histogram and probabilities*/
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            res[i][j]=0;
        }

        Pr[i]=0;
        Pf[i]=0;
    }

    //Find the center of each image
    if (R_M%2==1){
        cR.y=ceil((double)(R_M)/2);
    }
    else{
        cR.y=R_M/2;
    }


    if (R_N%2==1){
        cR.x=ceil((double)(R_N)/2);
    }
    else{
        cR.x=R_N/2;
    }


    if (F_M%2==1){
        cF.y=ceil((double)(F_M)/2);
    }
    else{
        cF.y=F_M/2;
    }


    if (F_N%2==1){
        cF.x=ceil((double)(F_N)/2);
    }
    else{
        cF.x=F_N/2;
    }



    /*Setting the values of the joint histogram*/
    for(i=0;i<F_N;i=i+step){
        for(j=0;j<F_M;j=j+step){

            /*Transform the coordinates of the Floating image  using the
            parameters a,e,f and the centers of the images*/
            TFpoint.x=cos(a)*(i+1-cF.x)- sin(a)*(j+1-cF.y)+cR.x-1+e;
            TFpoint.y=sin(a)*(i+1-cF.x)+ cos(a)*(j+1-cF.y)+cR.y-1+f;


            /*Check only poits of Floating image that when they are transformed in space
            they are inside the space of the Reference Image*/
            if (TFpoint.x>=0 && TFpoint.x<R_N && TFpoint.y>=0 && TFpoint.y<R_M){
                y=(unsigned int)F[j][i];
                q=r=0;

                /*First case: The transformed coordinates of the Floating image'ss pixel
                are integers*/
                if(floor(TFpoint.x) ==TFpoint.x && floor(TFpoint.y) ==TFpoint.y){


                    x=R[(unsigned int)TFpoint.y][(unsigned int)TFpoint.x];


                    res[x][y]=res[x][y]+1;
                }

                /*Second case: The transformed coordinate x of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.x) ==TFpoint.x){



                     if(floor(TFpoint.y)>=0){
                        q=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];



                     }

                     if((unsigned int)ceil(TFpoint.y)<R_M){
                        r=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];



                     }
                     x=roundf(q*(ceil(TFpoint.y)-TFpoint.y)+r*(TFpoint.y-floor(TFpoint.y)))↩
                         ;
                     res[x][y]=res[x][y]+1;
                }


                /*Third case: The transformed coordinate y of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.y) ==TFpoint.y){



                    if(floor(TFpoint.x)>=0){
                        q=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];


                    }

                    if((unsigned int)ceil(TFpoint.x)<R_N){
                        r=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];


                    }

                    x=roundf(q*(ceil(TFpoint.y)-TFpoint.y)+r*(TFpoint.y-floor(TFpoint.y)));
                res[x][y]=res[x][y]+1;
            }


            /*Fourth case: The transformed coordinates of the Floating image's pixel
            aren't integers*/
            else{
                x=0;


                if(floor(TFpoint.y)>=0 && floor(TFpoint.x)>=0){
                    q=R[(unsigned int)floor(TFpoint.y)][(unsigned int)floor(TFpoint.x)↩
                        ];


                }

                if(floor(TFpoint.y)>=0 && ceil(TFpoint.x)<R_N){
                    r=R[(unsigned int)floor(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];


                }
                x+=(ceil(TFpoint.y)-TFpoint.y)*(ceil(TFpoint.x)-TFpoint.x)*q+(ceil(↩
                    TFpoint.y)-TFpoint.y)*(TFpoint.x-floor(TFpoint.x))*r;
                q=r=0;

                if(floor(TFpoint.x)>0 && ceil(TFpoint.y)<R_M){
                    q=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)floor(TFpoint.x)];


                }

                if(ceil(TFpoint.y)<R_M && ceil(TFpoint.x)<R_N){
                    r=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];


                }
                x+=(TFpoint.y-floor(TFpoint.y))*(ceil(TFpoint.x)-TFpoint.x)*q+(TFpoint.↩
                    y-floor(TFpoint.y))*(TFpoint.x-floor(TFpoint.x))*r;
                x=roundf(x);
                res[x][y]=res[x][y]+1;
           }


        }

        if(x<0 || x>255){
            printf("x is %d\n",x);
            return 0;
        }

        if(y<0 || y>255){
            printf("y is %d\n",y);
            return 0;
        }
```

```
            }
        }

        for(i=0;i<256;i++){
            for(j=0;j<256;j++)
                sum+=res[i][j];
        }

        /*Normalization of joint histogram so tha the sum of its values is equal to 1*/

        for(i=0;i<256;i++){
            for(j=0;j<256;j++)
                res[i][j]=res[i][j]/sum;
        }


        /*Calculating marginal probabilities*/
        for(i=0;i<256;i++){
            for(j=0;j<256;j++){
                Pf[i]+=res[i][j];
                Pr[j]+=res[i][j];
            }
        }

        /*Calculcating marginal and joint entropy*/
        for(i=0;i<256;i++){
            if(Pr[i]>0){
                Hr+=-Pr[i]*log(Pr[i])/log(2.0);
            }
            if(Pf[i]>0)
                Hf+=-Pf[i]*log(Pf[i])/log(2.0);
            for(j=0;j<256;j++){
                if(res[i][j]>0){
                    Hrf+=-res[i][j]*log(res[i][j])/log(2.0);
                }
            }
        }

        //Calculation of mutual information;
        sum=Hr+Hf-Hrf;



        return -sum;

    }


    /*Calculation of Mutual information using pvi approximation.
    Instead of calculating the interpolated intensity in subpixel level,
    it distributes the contribution of the 4 nearest (8 in 3D) pixels to
    them.

    The variables are:
        input:
            arg[]: the parameters of the transformation operator
        output:
            Mutual information
        res:    the joint histogram
        Pr[i]:  probability of intensity i in Reference image
        Pf[i]:  probability of intensity i in Floating image
        Hf, Hr: marginal entropy of Floating and Reference image respectively
        Hrf:    joint entropy
        cF, cR: centers of Floating and Reference image respectively
        TFpoint: transformed coordinates of point in Floating image
    */
    float pvi_approx(float arg[]){
        unsigned int  i,j,x=1,y=1;

        float a,e,f,Pr[256],Pf[256],Hf=0,Hr=0,Hrf=0;
        float res[256][256];
        float sum=0;

        struct point TFpoint,cR,cF;
        FILE *f1,*f2,*f3;




        a=arg[0];//Rotation in radians

        e=arg[1];//Translation on x axis
        f=arg[2];//Translation on y axis


        //Initialization of joint histogram and marginal probabilities
        for(i=0;i<256;i++){
            for(j=0;j<256;j++){
                res[i][j]=0;
            }

            Pr[i]=0;
            Pf[i]=0;
        }


        /*Calculating the centre of each image*/
        if (R_M%2==1){
            cR.y=ceil((double)(R_M)/2);
        }
        else{
            cR.y=R_M/2;
        }


        if (R_N%2==1){
            cR.x=ceil((double)(R_N)/2);
        }
        else{
            cR.x=R_N/2;
        }


        if (F_M%2==1){
            cF.y=ceil((double)(F_M)/2);
        }
        else{
            cF.y=F_M/2;
        }


        if  (F_N%2==1){
            cF.x=ceil((double)(F_N)/2);
        }
        else{
            cF.x=F_N/2;
        }



        /*Setting the joint histogram*/
        for(i=0;i<F_N;i=i+step){
            for(j=0;j<F_M;j=j+step){

                /*Transform the coordinates of the Floating image  using the
                parameters a,e,f and the centers of the images*/
                TFpoint.x=cos(a)*(i+1-cF.x)- sin(a)*(j+1-cF.y)+cR.x-1+e;
                TFpoint.y=sin(a)*(i+1-cF.x)+ cos(a)*(j+1-cF.y)+cR.y-1+f;


                /*Check only poits of Floating image that when they are transformed in space
                they are inside the space of the Reference Image*/
                if  (TFpoint.x>=0 && TFpoint.x<R_N && TFpoint.y>=0 && TFpoint.y<R_M){
                    y=(unsigned int)F[j][i];

                    /*First case: The transformed coordinates of the Floating image'ss pixel
                    are integers*/
                    if(floor(TFpoint.x) ==TFpoint.x && floor(TFpoint.y) ==TFpoint.y){


                        x=R[(unsigned int)TFpoint.y][(unsigned int)TFpoint.x];


                        res[x][y]=res[x][y]+1;
                    }

                    /*Second case: The transformed coordinate x of the Floating image's pixel
                    is integer*/
                    else  if(floor(TFpoint.x) ==TFpoint.x){



                         if(floor(TFpoint.y)>=0){
                            x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];

                            res[x][y]=res[x][y]+(ceil(TFpoint.y)-TFpoint.y);

                         }

                         if((unsigned int)ceil(TFpoint.y)<R_M){
                            x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];

                            res[x][y]=res[x][y]+(TFpoint.y-floor(TFpoint.y));

                         }
                    }
            /*Third case: The transformed coordinate y of the Floating image's pixel
            is integer*/
            else if(floor(TFpoint.y) ==TFpoint.y){



                if(floor(TFpoint.x)>=0){
                    x=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];

                    res[x][y]=res[x][y]+(ceil(TFpoint.x)-TFpoint.x);
                }

                if((unsigned int)ceil(TFpoint.x)<R_N){
                    x=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];

                    res[x][y]=res[x][y]+(TFpoint.x-floor(TFpoint.x));
                }
            }

            /*Fourth case: The transformed coordinates of the Floating image's pixel
            aren't integers*/
            else{


                if(floor(TFpoint.y)>=0 && floor(TFpoint.x)>=0){
                    x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)floor(TFpoint.x)↩
                        ];

                    res[x][y]=res[x][y]+(ceil(TFpoint.y)-TFpoint.y)*(ceil(TFpoint.x)-↩
                        TFpoint.x);
                }

                if(floor(TFpoint.y)>=0 && ceil(TFpoint.x)<R_N){
                    x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                    res[x][y]=res[x][y]+(ceil(TFpoint.y)-TFpoint.y)*(TFpoint.x-floor(↩
                        TFpoint.x));
                }

                if(floor(TFpoint.x)>0 && ceil(TFpoint.y)<R_M){
                    x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)floor(TFpoint.x)];

                    res[x][y]=res[x][y]+(TFpoint.y-floor(TFpoint.y))*(ceil(TFpoint.x)-↩
                        TFpoint.x);
                }

                if(ceil(TFpoint.y)<R_M && ceil(TFpoint.x)<R_N){
                    x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                    res[x][y]=res[x][y]+(TFpoint.x-floor(TFpoint.x))*(TFpoint.y-floor(↩
                        TFpoint.y));
                }
           }


        }

        if(x<0 || x>255){
            printf("x is %d\n",x);
```

```
                return 0;
            }

            if(y<0 || y>255){
                printf("y is %d\n",y);
                return 0;
            }






        }
    }

    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            sum+=res[i][j];
    }

    //Normalizing the joint histogram so that the sum of its values is equal to 1
    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            res[i][j]=res[i][j]/sum;
    }



    //Calculating marginal probabilities
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            Pf[i]+=res[i][j];
            Pr[j]+=res[i][j];
        }
    }


    /*Calculating marginal and joint entropy*/
    for(i=0;i<256;i++){
        if(Pr[i]>0){
            Hr+=-Pr[i]*log(Pr[i])/log(2.0);
        }
        if(Pf[i]>0)
            Hf+=-Pf[i]*log(Pf[i])/log(2.0);
        for(j=0;j<256;j++){
            if(res[i][j]>0){
                Hrf+=-res[i][j]*log(res[i][j])/log(2.0);
            }
        }
    }

    //Calculation of mutual information
    sum=Hr+Hf-Hrf;



    return -sum;

}
```

Code of Extended Method

Now we have the extended method's code. The genetic algorithm is a slightly modified example created by Dennis Cormier and Sita Raghavan.

```
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <fstream>
# include <iomanip>
# include <cmath>
# include <ctime>
# include <cstring>

using namespace std;
//
//  Change any of these parameters to match your needs
//
#define F_M 256
#define F_N 256
#define R_M 256
#define R_N 256
#define PI 3.14159265

unsigned char **F,**R;
//float sum;


struct point{
    double x,y;
};


float roundf(float x){
    return floor(x + 0.5);
}
int step=1;
# define POPSIZE 100
# define MAXGENS 4000
# define NVARS 3
# define PXOVER 0.6
# define PMUTATION 0.95
//
//  Each GENOTYPE is a member of the population, with
//  gene: a string of variables,
//  fitness: the fitness
//  upper: the variable upper bounds,
//  lower: the variable lower bounds,
//  rfitness: the relative fitness,
//  cfitness: the cumulative fitness.
//
struct genotype
{
  float gene[NVARS];
  float fitness;
  float upper[NVARS];
  float lower[NVARS];
  float rfitness;
  float cfitness;
};

struct genotype population[POPSIZE+1];
struct genotype newpopulation[POPSIZE+1];

void crossover ( );
void elitist ( );
void evaluate ( );
void initialize ( string file_in_name );
void keep_the_best ( );
int main ( );
void mutate ( );
void r8_swap ( float *, float * );
float randval ( float, float );
void report ( int generation );
void selector ( );
void timestamp ( );
void Xover ( int, int );
void readImages();
float NN_approx(float arg[]);
float bilinear_approx(float arg[]);
float pvi_approx(float arg[]);
//****************************************************************************80

int main ( )

//****************************************************************************80
//
//  Purpose:
//
//    MAIN supervises the genetic algorithm.
//
//  Discussion:
//
//    Each generation involves selecting the best
//    members, performing crossover & mutation and then
//    evaluating the resulting population, until the terminating
//    condition is satisfied
//
//    This is a simple genetic algorithm implementation where the
//    evaluation function takes positive values only and the
//    fitness of an individual is the same as the value of the
//    objective function.
//
//  Modified:
//
//    29 December 2007
//
//  Parameters:
//
//    MAXGENS is the maximum number of generations.
//
//    NVARS is the number of problem variables.
//
//    PMUTATION is the probability of mutation.
//
```

```
//    POPSIZE is the population size.
//
//    PXOVER is the probability of crossover.
//
{
  int generation;
  int i;
  readImages();
  timestamp ( );
  float p[3];
  p[0]=-0.01923;
  p[1]=9.3212;
  p[2]=17.2945;
  cout<<"this is "<<fun6(p);
  cout << "\n";
  cout << "SIMPLE_GA:\n";
  cout << "  C++ version\n";
  cout << "\n";
  cout << "  A simple example of a genetic algorithm.\n";

  cout << "\n";
  cout << " Generation  Best  Average  Standard \n";
  cout << " number      value fitness  deviation \n";
  cout << "\n";

  initialize ( "simple_ga_input.txt" );

  evaluate ( );

  keep_the_best () ;

  for ( generation =  0; generation < MAXGENS; generation++ )
  {
    selector ( );
    crossover ( );
    mutate ( );
    report ( generation );
    evaluate ( );
    elitist ( );
  }

  cout << "\n";
  cout << "\n";
  cout << "Simulation completed.\n";

  cout << "\n";
  cout << "Best member:\n";
  cout << "\n";

  for ( i = 0; i < NVARS; i++ )
  {
    cout << "var(" << i << ") = " << population[POPSIZE].gene[i] << "\n";
  }

  cout << "\n";
  cout << "\n";
  cout << "Best fitness = " << population[POPSIZE].fitness << "\n";
//
//  Terminate.
```

```
//
  cout << "\n";
  cout << "SIMPLE_GA:\n";
  cout << "  Normal end of execution.\n";

  cout << "\n";
  timestamp ( );
  system("PAUSE");
  return 0;
}
//****************************************************************************80
void readImages(){
    unsigned int  i,j,x=1,y=1;
    FILE *f1,*f2,*f3;

    /*F_M=134;
    F_N=175;
    R_M=134;
    R_N=175;*/

    F=(unsigned char**)malloc(F_M*sizeof(unsigned char*));
    R=(unsigned char**)malloc(R_M*sizeof(unsigned char*));




    for(i=0;i<F_M;i++){
        F[i]=(unsigned char*)malloc(F_N*sizeof(unsigned char));
        R[i]=(unsigned char*)malloc(R_N*sizeof(unsigned char));
    }


    f1=fopen("myFile3.txt","r");
    f2=fopen("myFile4.txt","r");
    //f3=fopen("myPoints.txt","a+");

    if(f1==NULL || f2==NULL ){
        printf("problem1\n");
        return;
    }
    i=j=0;
    //printf("problem\n");
    while(!feof(f1)){
        fscanf(f1,"%uc",&F[i][j]);
        j++;
        if(j==F_N){
            j=0;
            i++;
        }
    }
    fclose(f1);

    i=j=0;
    while(!feof(f2)){
        fscanf(f2,"%uc",&R[i][j]);
        j++;
        if(j==R_N){
            j=0;
            i++;
        }
    }
    fclose(f2);
}

/*Calculation of Mutual Information using Nearest Neighbour
approximation. It is a crude way to calculate Mutual Information,
prone to give a false image of the similarity of the Images

The variables are:
    input:
        arg[]: the parameters of the transformation operator
    output:
        Mutual information
    res:    the joint histogram
    Pr[i]:  probability of intensity i in Reference image
    Pf[i]:  probability of intensity i in Floating image
    Hf, Hr: marginal entropy of Floating and Reference image respectively
    Hrf:    joint entropy
    cF, cR: centers of Floating and Reference image respectively
    TFpoint: transformed coordinates of point in Floating image
*/

float NN_approx(float arg[]){
    unsigned int  i,j,x=1,y=1,xb,yb;


    float a,e,f,Pr[256],Pf[256],Hf=0,Hr=0,Hrf=0;
    float res[256][256];
    float sum=0;

    struct point TFpoint,cR,cF;

    a=arg[0];//Rotation in radians

    e=arg[1];//Translation in x axis
    f=arg[2];//Translation in y axis

    //Initialize joint histogram and probability function
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            res[i][j]=0;
        }

        Pr[i]=0;
        Pf[i]=0;
    }

    //Find centres of Images
    if (R_M%2==1){
        cR.y=ceil((double)(R_M)/2);
    }
    else{
        cR.y=R_M/2;
    }


    if (R_N%2==1){
        cR.x=ceil((double)(R_N)/2);
    }
    else{
        cR.x=R_N/2;
    }


    if (F_M%2==1){
        cF.y=ceil((double)(F_M)/2);
    }
    else{
        cF.y=F_M/2;
    }


    if (F_N%2==1){
        cF.x=ceil((double)(F_N)/2);
    }
    else{
        cF.x=F_N/2;
    }



    /*Making of the joint histgram*/
    for(i=0;i<F_N;i=i+step){
        for(j=0;j<F_M;j=j+step){

            /*Transform the coordinates of the Floating image  using the
            parameters a,e,f and the centers of the images*/
            TFpoint.x=cos(a)*(i+1-cF.x)- sin(a)*(j+1-cF.y)+cR.x-1+e;
            TFpoint.y=sin(a)*(i+1-cF.x)+ cos(a)*(j+1-cF.y)+cR.y-1+f;

            /*Check only poits of Floating image that when they are transformed in space
            they are inside the space of the Reference Image*/
            if (TFpoint.x>=0 && TFpoint.x<R_N && TFpoint.y>=0 && TFpoint.y<R_M){
                y=(unsigned int)F[j][i];


                /*First case: The transformed coordinates of the Floating image'ss pixel
                are integers*/
                if(floor(TFpoint.x) ==TFpoint.x && floor(TFpoint.y) ==TFpoint.y){

                    x=R[(unsigned int)TFpoint.y][(unsigned int)TFpoint.x];

                    res[x][y]=res[x][y]+1;
                }

                /*Second case: The transformed coordinate x of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.x) ==TFpoint.x){


                    if(floor(TFpoint.y)<0){

                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];
                        res[x][y]=res[x][y]+1;
                    }
                    else if(ceil(TFpoint.y)>=R_M){
```

```
                    x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];
                    res[x][y]=res[x][y]+1;
                }
                else if(fabs(floor(TFpoint.y)-TFpoint.y)<fabs(ceil(TFpoint.y)-TFpoint.y↩
                    )){
                    x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];

                    res[x][y]=res[x][y]+1;

                }
                else{
                    x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];

                    res[x][y]=res[x][y]+1;

                 }
            }

            /*Third case: The transformed coordinate y of the Floating image's pixel
            is integer*/
            else if(floor(TFpoint.y) ==TFpoint.y){



                if(floor(TFpoint.x)<0){

                    x=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];
                    res[x][y]=res[x][y]+1;
                }
                else if(ceil(TFpoint.x)>=R_N){

                    x=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];
                    res[x][y]=res[x][y]+1;
                }
                else if(fabs(floor(TFpoint.x)-TFpoint.x)<fabs(ceil(TFpoint.x)-TFpoint.x↩
                    )){
                    x=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];

                    res[x][y]=res[x][y]+1;
                }
                else{
                    x=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];

                    res[x][y]=res[x][y]+1;
                }
            }

            /*Fourth case: The transformed coordinates of the Floating image's pixel
            aren't integers*/
            else{

                if(floor(TFpoint.x)<0){

                    xb=1;
                }
                else if(ceil(TFpoint.x)>=R_N){

                    xb=0;
                }
            else if(fabs(floor(TFpoint.x)−TFpoint.x)<fabs(ceil(TFpoint.x)−TFpoint.x↩
                )){
                xb=0;
            }
            else{
                xb=1;
            }


            if(floor(TFpoint.y)<0){

                yb=1;
            }
            else if(ceil(TFpoint.y)>=R_M){

                yb=0;
            }
            else if(fabs(floor(TFpoint.y)−TFpoint.y)<fabs(ceil(TFpoint.y)−TFpoint.y↩
                )){
                yb=0;
            }
            else{
                yb=1;
            }



            if(xb==0 && yb==0){
                x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)floor(TFpoint.x)↩
                    ];

                res[x][y]=res[x][y]+1;
            }
            else if(xb==1 && yb==0){
                x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                res[x][y]=res[x][y]+1;
            }
            else if(xb==0 && yb==1){
                x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)floor(TFpoint.x)];

                res[x][y]=res[x][y]+1;
            }

            if(xb==1 && yb==1){
                x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                res[x][y]=res[x][y]+1;
            }
        }


    }

    if(x<0 || x>255){
        printf("x is %d\n",x);
        return 0;
    }
```

```
                if(y<0 || y>255){
                    printf("y is %d\n",y);
                    return 0;
                }







            }
        }

        for(i=0;i<256;i++){
            for(j=0;j<256;j++)
                sum+=res[i][j];
        }

        /*Normalization of joint histogram so that the sum of its values is equal to 1*/
        for(i=0;i<256;i++){
            for(j=0;j<256;j++)
                res[i][j]=res[i][j]/sum;
        }


        /*Calculation of probabilities*/
        for(i=0;i<256;i++){
            for(j=0;j<256;j++){
                Pf[i]+=res[i][j];
                Pr[j]+=res[i][j];
            }
        }

        /*Calculation of marginal and joint entropy*/
        for(i=0;i<256;i++){
            if(Pr[i]>0){
                Hr+=-Pr[i]*log(Pr[i])/log(2.0);
            }
            if(Pf[i]>0)
                Hf+=-Pf[i]*log(Pf[i])/log(2.0);
            for(j=0;j<256;j++){
                if(res[i][j]>0){
                    Hrf+=-res[i][j]*log(res[i][j])/log(2.0);
                }
            }
        }

        /*Calculation of mutual information*/
        sum=Hr+Hf-Hrf;


        return -sum;

}


/*Calculation of Mutual information using bilinear interpolation approximation.
It is more accurate than Nearest Neighbour approximation, but tends ti introduce
in subpixel level new intensities that don't exist in the image. It is less
prone to give a false image of the similarity of the Images
```

```
The variables are:
    input:
        arg[]: the parameters of the transformation operator
    output:
        Mutual information
    res:    the joint histogram
    Pr[i]:  probability of intensity i in Reference image
    Pf[i]:  probability of intensity i in Floating image
    Hf, Hr: marginal entropy of Floating and Reference image respectively
    Hrf:    joint entropy
    cF, cR: centers of Floating and Reference image respectively
    TFpoint: transformed coordinates of point in Floating image
*/
float bilinear_approx(float arg[]){
    unsigned int  i,j,x=1,y=1;

    float a,e,f,Pr[256],Pf[256],Hf=0,Hr=0,Hrf=0,q,r;
    float res[256][256];
    float sum=0;

    struct point TFpoint,cR,cF;
    FILE *f1,*f2,*f3;




    a=arg[0];//Rotation in radians

    e=arg[1];//Translation on x axis
    f=arg[2];//Translation on y axis

    /*Initialize joint histogram and probabilities*/
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            res[i][j]=0;
        }

        Pr[i]=0;
        Pf[i]=0;
    }

    //Find the center of each image
    if (R_M%2==1){
        cR.y=ceil((double)(R_M)/2);
    }
    else{
        cR.y=R_M/2;
    }


    if (R_N%2==1){
        cR.x=ceil((double)(R_N)/2);
    }
    else{
        cR.x=R_N/2;
    }
    if (F_M%2==1){
        cF.y=ceil((double)(F_M)/2);
    }
    else{
        cF.y=F_M/2;
    }


    if (F_N%2==1){
        cF.x=ceil((double)(F_N)/2);
    }
    else{
        cF.x=F_N/2;
    }



    /*Setting the values of the joint histogram*/
    for(i=0;i<F_N;i=i+step){
        for(j=0;j<F_M;j=j+step){

            /*Transform the coordinates of the Floating image  using the
            parameters a,e,f and the centers of the images*/
            TFpoint.x=cos(a)*(i+1-cF.x)- sin(a)*(j+1-cF.y)+cR.x-1+e;
            TFpoint.y=sin(a)*(i+1-cF.x)+ cos(a)*(j+1-cF.y)+cR.y-1+f;


            /*Check only poits of Floating image that when they are transformed in space
            they are inside the space of the Reference Image*/
            if (TFpoint.x>=0 && TFpoint.x<R_N && TFpoint.y>=0 && TFpoint.y<R_M){
                y=(unsigned int)F[j][i];
                q=r=0;

                /*First case: The transformed coordinates of the Floating image'ss pixel
                are integers*/
                if(floor(TFpoint.x) ==TFpoint.x && floor(TFpoint.y) ==TFpoint.y){


                    x=R[(unsigned int)TFpoint.y][(unsigned int)TFpoint.x];


                    res[x][y]=res[x][y]+1;
                }

                /*Second case: The transformed coordinate x of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.x) ==TFpoint.x){



                     if(floor(TFpoint.y)>=0){
                        q=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];



                     }

                     if((unsigned int)ceil(TFpoint.y)<R_M){
                        r=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];
            }
            x=roundf(q*(ceil(TFpoint.y)-TFpoint.y)+r*(TFpoint.y-floor(TFpoint.y)))↩
                ;
            res[x][y]=res[x][y]+1;
        }


        /*Third case: The transformed coordinate y of the Floating image's pixel
        is integer*/
        else if(floor(TFpoint.y) ==TFpoint.y){



            if(floor(TFpoint.x)>=0){
                q=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];


            }

            if((unsigned int)ceil(TFpoint.x)<R_N){
                r=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];


            }

            x=roundf(q*(ceil(TFpoint.y)-TFpoint.y)+r*(TFpoint.y-floor(TFpoint.y)));
            res[x][y]=res[x][y]+1;
        }


        /*Fourth case: The transformed coordinates of the Floating image's pixel
        aren't integers*/
        else{
            x=0;


            if(floor(TFpoint.y)>=0 && floor(TFpoint.x)>=0){
                q=R[(unsigned int)floor(TFpoint.y)][(unsigned int)floor(TFpoint.x)↩
                    ];


            }

            if(floor(TFpoint.y)>=0 && ceil(TFpoint.x)<R_N){
                r=R[(unsigned int)floor(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];


            }
            x+=(ceil(TFpoint.y)-TFpoint.y)*(ceil(TFpoint.x)-TFpoint.x)*q+(ceil(↩
                TFpoint.y)-TFpoint.y)*(TFpoint.x-floor(TFpoint.x))*r;
            q=r=0;

            if(floor(TFpoint.x)>0 && ceil(TFpoint.y)<R_M){
                q=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)floor(TFpoint.x)];
```

```
                    }

                    if(ceil(TFpoint.y)<R_M && ceil(TFpoint.x)<R_N){
                        r=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];


                    }
                    x+=(TFpoint.y-floor(TFpoint.y))*(ceil(TFpoint.x)-TFpoint.x)*q+(TFpoint.↩
                        y-floor(TFpoint.y))*(TFpoint.x-floor(TFpoint.x))*r;
                    x=roundf(x);
                    res[x][y]=res[x][y]+1;
                }


            }

            if(x<0 || x>255){
                printf("x is %d\n",x);
                return 0;
            }

            if(y<0 || y>255){
                printf("y is %d\n",y);
                return 0;
            }





        }
    }

    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            sum+=res[i][j];
    }

    /*Normalization of joint histogram so tha the sum of its values is equal to 1*/

    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            res[i][j]=res[i][j]/sum;
    }


    /*Calculating marginal probabilities*/
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            Pf[i]+=res[i][j];
            Pr[j]+=res[i][j];
        }
    }

    /*Calculcating marginal and joint entropy*/
    for(i=0;i<256;i++){
        if(Pr[i]>0){
            Hr+=-Pr[i]*log(Pr[i])/log(2.0);
        }
        if(Pf[i]>0)
            Hf+=-Pf[i]*log(Pf[i])/log(2.0);
        for(j=0;j<256;j++){
            if(res[i][j]>0){
                Hrf+=-res[i][j]*log(res[i][j])/log(2.0);
            }
        }
    }

    //Calculation of mutual information;
    sum=Hr+Hf-Hrf;



    return -sum;

}


/*Calculation of Mutual information using pvi approximation.
Instead of calculating the interpolated intensity in subpixel level,
it distributes the contribution of the 4 nearest (8 in 3D) pixels to
them.

The variables are:
    input:
        arg[]: the parameters of the transformation operator
    output:
        Mutual information
    res:    the joint histogram
    Pr[i]:  probability of intensity i in Reference image
    Pf[i]:  probability of intensity i in Floating image
    Hf, Hr: marginal entropy of Floating and Reference image respectively
    Hrf:    joint entropy
    cF, cR: centers of Floating and Reference image respectively
    TFpoint: transformed coordinates of point in Floating image
*/
float pvi_approx(float arg[]){
    unsigned int  i,j,x=1,y=1;

    float a,e,f,Pr[256],Pf[256],Hf=0,Hr=0,Hrf=0;
    float res[256][256];
    float sum=0;

    struct point TFpoint,cR,cF;
    FILE *f1,*f2,*f3;




    a=arg[0];//Rotation in radians

    e=arg[1];//Translation on x axis
    f=arg[2];//Translation on y axis


    //Initialization of joint histogram and marginal probabilities
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            res[i][j]=0;
        }

        Pr[i]=0;
        Pf[i]=0;
    }


    /*Calculating the centre of each image*/
    if (R_M%2==1){
        cR.y=ceil((double)(R_M)/2);
    }
    else{
        cR.y=R_M/2;
    }


    if (R_N%2==1){
        cR.x=ceil((double)(R_N)/2);
    }
    else{
        cR.x=R_N/2;
    }


    if (F_M%2==1){
        cF.y=ceil((double)(F_M)/2);
    }
    else{
        cF.y=F_M/2;
    }


    if (F_N%2==1){
        cF.x=ceil((double)(F_N)/2);
    }
    else{
        cF.x=F_N/2;
    }



    /*Setting the joint histogram*/
    for(i=0;i<F_N;i=i+step){
        for(j=0;j<F_M;j=j+step){

            /*Transform the coordinates of the Floating image  using the
            parameters a,e,f and the centers of the images*/
            TFpoint.x=cos(a)*(i+1-cF.x)- sin(a)*(j+1-cF.y)+cR.x-1+e;
            TFpoint.y=sin(a)*(i+1-cF.x)+ cos(a)*(j+1-cF.y)+cR.y-1+f;


            /*Check only poits of Floating image that when they are transformed in space
            they are inside the space of the Reference Image*/
            if (TFpoint.x>=0 && TFpoint.x<R_N && TFpoint.y>=0 && TFpoint.y<R_M){
                y=(unsigned int)F[j][i];

                /*First case: The transformed coordinates of the Floating image'ss pixel
                are integers*/
                if(floor(TFpoint.x) ==TFpoint.x && floor(TFpoint.y) ==TFpoint.y){


                    x=R[(unsigned int)TFpoint.y][(unsigned int)TFpoint.x];


                    res[x][y]=res[x][y]+1;
                }

                /*Second case: The transformed coordinate x of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.x) ==TFpoint.x){



                     if(floor(TFpoint.y)>=0){
                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)TFpoint.x];

                        res[x][y]=res[x][y]+(ceil(TFpoint.y)-TFpoint.y);

                     }

                     if((unsigned int)ceil(TFpoint.y)<R_M){
                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)TFpoint.x];

                        res[x][y]=res[x][y]+(TFpoint.y-floor(TFpoint.y));

                     }
                }
                /*Third case: The transformed coordinate y of the Floating image's pixel
                is integer*/
                else if(floor(TFpoint.y) ==TFpoint.y){



                    if(floor(TFpoint.x)>=0){
                        x=R[(unsigned int)TFpoint.y][(unsigned int)floor(TFpoint.x)];

                        res[x][y]=res[x][y]+(ceil(TFpoint.x)-TFpoint.x);
                    }

                    if((unsigned int)ceil(TFpoint.x)<R_N){
                        x=R[(unsigned int)TFpoint.y][(unsigned int)ceil(TFpoint.x)];

                        res[x][y]=res[x][y]+(TFpoint.x-floor(TFpoint.x));
                    }
                }

                /*Fourth case: The transformed coordinates of the Floating image's pixel
                aren't integers*/
                else{


                    if(floor(TFpoint.y)>=0 && floor(TFpoint.x)>=0){
                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)floor(TFpoint.x)↩
                            ];

                        res[x][y]=res[x][y]+(ceil(TFpoint.y)-TFpoint.y)*(ceil(TFpoint.x)-↩
                            TFpoint.x);
                    }

                    if(floor(TFpoint.y)>=0 && ceil(TFpoint.x)<R_N){
                        x=R[(unsigned int)floor(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                        res[x][y]=res[x][y]+(ceil(TFpoint.y)-TFpoint.y)*(TFpoint.x-floor(↩
                            TFpoint.x));
                    }

                    if(floor(TFpoint.x)>0 && ceil(TFpoint.y)<R_M){
                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)floor(TFpoint.x)];

                        res[x][y]=res[x][y]+(TFpoint.y-floor(TFpoint.y))*(ceil(TFpoint.x)-↩
                            TFpoint.x);
                    }

                    if(ceil(TFpoint.y)<R_M && ceil(TFpoint.x)<R_N){
                        x=R[(unsigned int)ceil(TFpoint.y)][(unsigned int)ceil(TFpoint.x)];

                        res[x][y]=res[x][y]+(TFpoint.x-floor(TFpoint.x))*(TFpoint.y-floor(↩
                            TFpoint.y));
                    }
                }


            }

            if(x<0 || x>255){
                printf("x is %d\n",x);
                return 0;
            }

            if(y<0 || y>255){
                printf("y is %d\n",y);
                return 0;
            }





        }
    }

    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            sum+=res[i][j];
    }

    //Normalizing the joint histogram so that the sum of its values is equal to 1
    for(i=0;i<256;i++){
        for(j=0;j<256;j++)
            res[i][j]=res[i][j]/sum;
    }



    //Calculating marginal probabilities
    for(i=0;i<256;i++){
        for(j=0;j<256;j++){
            Pf[i]+=res[i][j];
            Pr[j]+=res[i][j];
        }
    }


    /*Calculating marginal and joint entropy*/
    for(i=0;i<256;i++){
        if(Pr[i]>0){
            Hr+=-Pr[i]*log(Pr[i])/log(2.0);
        }
        if(Pf[i]>0)
            Hf+=-Pf[i]*log(Pf[i])/log(2.0);
        for(j=0;j<256;j++){
            if(res[i][j]>0){
                Hrf+=-res[i][j]*log(res[i][j])/log(2.0);
            }
        }
    }

    //Calculation of mutual information
    sum=Hr+Hf-Hrf;



    return -sum;

}

void crossover ( )

//****************************************************************************80
//
//  Purpose:
//
//    CROSSOVER selects two parents for the single point crossover.
//
//  Modified:
//
//    29 December 2007
//
//  Local parameters:
//
//    Local, int FIRST, is a count of the number of members chosen.
//
{
  int mem;
  int one;
  int first = 0;
  float x;

  for ( mem = 0; mem < POPSIZE; ++mem )
  {
    x = ( rand ( ) % 1000 ) / 1000.0;

    if ( x < PXOVER )
    {
      ++first;
      if ( first % 2 == 0 )
      {
        Xover ( one, mem );
      }
      else
      {
        one = mem;
      }

    }
  }
  return;
}
//****************************************************************************80

void elitist ( )

//****************************************************************************80
//
//  Purpose:
//
//    ELITIST stores the best member of the previous generation.
//
//  Discussion:
//
//    The best member of the previous generation is stored as
//    the last in the array. If the best member of the current
//    generation is worse then the best member of the previous
//    generation, the latter one would replace the worst member
//    of the current population.
//
//  Modified:
//
//    29 December 2007
//
//  Local parameters:
//
//    Local, float BEST, the best fitness value.
//
//    Local, float WORST, the worst fitness value.
//
{
  int i;
  float best;
  int best_mem;
  float worst;
  int worst_mem;

  best = population[0].fitness;
  worst = population[0].fitness;

  for ( i = 0; i < POPSIZE - 1; ++i )
  {
    if ( population[i].fitness > population[i+1].fitness )
    {

      if ( best <= population[i].fitness )
      {
        best = population[i].fitness;
        best_mem = i;
      }

      if ( population[i+1].fitness <= worst )
      {
        worst = population[i+1].fitness;
        worst_mem = i + 1;
      }

    }
    else
    {

      if ( population[i].fitness <= worst )
      {
        worst = population[i].fitness;
        worst_mem = i;
      }

      if ( best <= population[i+1].fitness )
      {
        best = population[i+1].fitness;
        best_mem = i + 1;
      }

    }

  }
//
//  If the best individual from the new population is better than
//  the best individual from the previous population, then
//  copy the best from the new population; else replace the
//  worst individual from the current population with the
//  best one from the previous generation
//
  if ( best >= population[POPSIZE].fitness )
  {
    for ( i = 0; i < NVARS; i++ )
    {
      population[POPSIZE].gene[i] = population[best_mem].gene[i];
    }
    population[POPSIZE].fitness = population[best_mem].fitness;
  }
  else
  {
    for ( i = 0; i < NVARS; i++ )
    {
      population[worst_mem].gene[i] = population[POPSIZE].gene[i];
    }
    population[worst_mem].fitness = population[POPSIZE].fitness;
  }

  return;

}
//****************************************************************************80

void evaluate ( )
```

```
//****************************************************************************80
//
//  Purpose:
//
//    EVALUATE implements the user-defined valuation function
//
//  Discussion:
//
//    Each time this is changed, the code has to be recompiled.
//    The current function is:  x[1]^2-x[1]*x[2]+x[3]
//
//  Modified:
//
//    29 December 2007
//
{
  int member;
  int i;
  float x[NVARS+1];
  float p[NVARS];

  for ( member = 0; member < POPSIZE; member++ )
  {
    for ( i = 0; i < NVARS; i++ )
    {
      x[i+1] =p[i]=population[member].gene[i];
    }
    //population[member].fitness = ( x[1] * x[1] ) - ( x[1] * x[2] ) + x[3];
    population[member].fitness = fun6(p);
  }
  return;
}
//****************************************************************************80

void initialize ( string file_in_name )

//****************************************************************************80
//
//  Purpose:
//
//    INITIALIZE initializes the genes within the variables bounds.
//
//  Discussion:
//
//    It also initializes (to zero) all fitness values for each
//    member of the population. It reads upper and lower bounds
//    of each variable from the input file `gadata.txt'. It
//    randomly generates values between these bounds for each
//    gene of each genotype in the population. The format of
//    the input file `gadata.txt' is
//
//      var1_lower_bound var1_upper bound
//      var2_lower_bound var2_upper bound ...
//
//  Modified:
//
//    29 December 2007
//
```

```
{
  ifstream file_in;
  int i;
  int j;
  float lbound;
  float ubound;

  file_in.open ( file_in_name.c_str ( ) );

  if ( !file_in )
  {
    cout << "\n";
    cout << "Initialize - Fatal error!\n";
    cout << "  Cannot open the input file!\n";
    exit ( 1 );
  }
//
//  Initialize variables within the bounds
//
  for ( i = 0; i < NVARS; i++ )
  {
    file_in >> lbound >> ubound;

    for ( j = 0; j < POPSIZE; j++ )
    {
      population[j].fitness = 0;
      population[j].rfitness = 0;
      population[j].cfitness = 0;
      population[j].lower[i] = lbound;
      population[j].upper[i]= ubound;
      population[j].gene[i] = randval ( population[j].lower[i],
        population[j].upper[i] );
    }
  }

  file_in.close ( );

  return;
}
//****************************************************************************80

void keep_the_best ( )

//****************************************************************************80
//
//  Purpose:
//
//    KEEP_THE_BEST keeps track of the best member of the population.
//
//  Discussion:
//
//    Note that the last entry in the array Population holds a
//    copy of the best individual.
//
//  Modified:
//
//    29 December 2007
//
//  Local parameters:
```

```
//
//    Local, int CUR_BEST, the index of the best individual.
//
{
  int cur_best;
  int mem;
  int i;

  cur_best = 0;

  for ( mem = 0; mem < POPSIZE; mem++ )
  {
    if ( population[mem].fitness > population[POPSIZE].fitness )
    {
      cur_best = mem;
      population[POPSIZE].fitness = population[mem].fitness;
    }
  }
//
//  Once the best member in the population is found, copy the genes.
//
  for ( i = 0; i < NVARS; i++ )
  {
    population[POPSIZE].gene[i] = population[cur_best].gene[i];
  }

  return;
}
//****************************************************************************80

void mutate ( )

//****************************************************************************80
//
//  Purpose:
//
//    MUTATE performs a random uniform mutation.
//
//  Discussion:
//
//    A variable selected for mutation is replaced by a random value
//    between the lower and upper bounds of this variable.
//
//  Modified:
//
//    29 December 2007
//
{
  float hbound;
  int i;
  int j;
  float lbound;
  float x;

  for ( i = 0; i < POPSIZE; i++ )
  {
    for ( j = 0; j < NVARS; j++ )
    {
      x = rand ( ) % 1000 / 1000.0;
//
//  Find the bounds on the variable to be mutated
//
      if ( x < PMUTATION )
      {
        lbound = population[i].lower[j];
        hbound = population[i].upper[j];
        population[i].gene[j] = randval ( lbound, hbound );
      }
    }
  }

  return;
}
//****************************************************************************80

void r8_swap ( float *x, float *y )

//****************************************************************************80
//
//  Purpose:
//
//    R8_SWAP swaps two R8's.
//
//  Modified:
//
//    29 December 2007
//
{
  float temp;

  temp = *x;
  *x = *y;
  *y = temp;

  return;
}
//****************************************************************************80

float randval ( float low, float high )

//****************************************************************************80
//
//  Purpose:
//
//    RANDVAL generates a random value within bounds.
//
//  Modified:
//
//    29 December 2007
//
{
  float val;

  val = ( ( float ) ( rand() % 1000 ) / 1000.0 )
    * ( high - low ) + low;

  return ( val );
}
```

```
//****************************************************************************80

void report ( int generation )

//****************************************************************************80
//
//  Purpose:
//
//    REPORT reports progress of the simulation.
//
//  Modified:
//
//    29 December 2007
//
//  Local parameters:
//
//    Local, float avg, the average population fitness.
//
//    Local, best_val, the best population fitness.
//
//    Local, float square_sum, square of sum for std calc.
//
//    Local, float stddev, standard deviation of population fitness.
//
//    Local, float sum, the total population fitness.
//
//    Local, float sum_square, sum of squares for std calc.
//
{
  float avg;
  float best_val;
  int i;
  float square_sum;
  float stddev;
  float sum;
  float sum_square;

  sum = 0.0;
  sum_square = 0.0;

  for ( i = 0; i < POPSIZE; i++ )
  {
    sum = sum + population[i].fitness;
    sum_square = sum_square + population[i].fitness * population[i].fitness;
  }

  avg = sum / ( float ) POPSIZE;
  square_sum = avg * avg * POPSIZE;
  stddev = sqrt ( ( sum_square - square_sum ) / ( POPSIZE - 1 ) );
  best_val = population[POPSIZE].fitness;

  cout << "  " << setw(8) << generation
       << " " << best_val
       << " " << avg
       << " " << stddev << " "<< population[POPSIZE].gene[0]<<" " << population[POPSIZE].↩
           gene[1]<< " "<< population[POPSIZE].gene[2] <<"\n";

  return;
}
```

```
//****************************************************************************80

void selector ( )

//****************************************************************************80
//
//  Purpose:
//
//    SELECTOR is the selection function.
//
//  Discussion:
//
//    Standard proportional selection for
//    maximization problems incorporating elitist model - makes
//    sure that the best member survives
//
//  Modified:
//
//    29 December 2007
//
{
  int i;
  int j;
  int mem;
  float p;
  float sum = 0;
//
//  Find total fitness of the population
//
  for ( mem = 0; mem < POPSIZE; mem++ )
  {
    sum = sum + population[mem].fitness;
  }
//
//  Calculate the relative fitness.
//
  for ( mem = 0; mem < POPSIZE; mem++ )
  {
    population[mem].rfitness = population[mem].fitness / sum;
  }
  population[0].cfitness = population[0].rfitness;
//
//  Calculate the cumulative fitness.
//
  for ( mem = 1; mem < POPSIZE; mem++ )
  {
    population[mem].cfitness = population[mem-1].cfitness +
      population[mem].rfitness;
  }
//
//  Select survivors using cumulative fitness.
//
  for ( i = 0; i < POPSIZE; i++ )
  {
    p = rand() % 1000 / 1000.0;
    if (p < population[0].cfitness)
    {
      newpopulation[i] = population[0];
    }
    else
    {
      for ( j = 0; j < POPSIZE; j++ )
      {
        if ( p >= population[j].cfitness && p < population[j+1].cfitness )
        {
          newpopulation[i] = population[j+1];
        }
      }
    }
  }
//
//  Once a new population is created, copy it back
//
  for ( i = 0; i < POPSIZE; i++ )
  {
    population[i] = newpopulation[i];
  }

  return;
}
//****************************************************************************80

void timestamp ( )

//****************************************************************************80
//
//  Purpose:
//
//    TIMESTAMP prints the current YMDHMS date as a time stamp.
//
//  Example:
//
//    May 31 2001 09:45:54 AM
//
//  Modified:
//
//    04 October 2003
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    None
//
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );
  cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}
//****************************************************************************80

void Xover ( int one, int two )

//****************************************************************************80
//
//  Purpose:
//
//    XOVER performs crossover of the two selected parents.
//
//  Modified:
//
//    29 December 2007
//
//  Local parameters:
//
//    Local, int point, the crossover point.
//
{
  int i;
  int point;
//
//  Select the crossover point.
//
  if ( 1 < NVARS )
  {

    if ( NVARS == 2 )
    {
      point = 1;
    }
    else
    {
      point = ( rand ( ) % ( NVARS - 1 ) ) + 1;
    }

    for ( i = 0; i < point; i++ )
    {
      r8_swap ( &population[one].gene[i], &population[two].gene[i] );
    }

  }
  return;
}
```

Bibliography

[1] Frederik Maes, Andre Collignon, Dirk Vandermeulen, Guy Marchal, and Paul Suetens *Multimodality Image Registration by Maximization of Mutual Information.* IEEE Trans. Image Processing, vol. 20, no. 8, pp. 2378-2386, 2011.

[2] Richard P. Brent *Algorithms for minimization without Derivatives* . Englewood Cliffs, N.J.: Prentice-Hall, 1972.

[3] Powell, M. J. D *An efficient method for finding the minimum of a function of several variables without calculating derivatives.* Computer Journal 7 (2): 155–162, 1964.

[4] Thomas M. Cover and Joy A. Thomas *Elements of Information Theory, 2nd Edition*, 2006.

[5] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang *FSIM: A Feature Similarity Index for Image Quality Assessment.* IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 16, NO. 2, APRIL 1997.

[6] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli *Image Quality Assessment: From Error Visibility to Structural Similarity.* IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 13, NO. 4, APRIL 2004

[7] Mehul P. Sampat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik and Mia K. Markey *Complex Wavelet Structural Similarity: A New Image Similarity Index.* IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 18, NO. 11, NOVEMBER 2009

[8] Dirk Thierens and David Goldberg, *Convergence Models of Genetic Algorithm Selection Schemes.* International Conference on Evolutionary Computation The Third Conference on Parallel Problem Solving from Nature Jerusalem, Israel, October 9–14, 1994 Proceedings

[9] Miller, Brad L.; Goldberg, David E., *Genetic Algorithms, Tournament Selection, and the Effects of Noise.* Complex Systems 9 (1995) 193- 212.

[10] Butz, Martin V.; Sastry, Kumara; Goldberg, David E.., *Tournament Selection in XCS.* Complex Systems 9 (1995) 193- 212.

[11] H. Muhlenbein, D. Schlierkamp-Voosen, *Predictive Models for the Breeder Genetic Algorithm.* Journal Evolutionary Computation, Volume 1 Issue 1, Spring 1993 Pages 25-49.

[12] Baker, J.E., *Reducing bias and inefficiency in the selection algorithm.* Genetic algorithms and their applications : proceedings of the second International Conference on Genetic Algorithms, July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA.

[13] John H. Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan. 1975