MOTION CORRECTION OF PET/CT IMAGES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Juan Antonio Kim Hoo Chong Chie

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2017

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Paul Salama, Co-Chair

Department of Electrical and Computer Engineering

Dr. Paul Territo, Co-Chair

Department of Radiology and Imaging Sciences, School of Medicine

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Maher Rizkalla

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of the Department Graduate Program

For my parents,

Santiago Chong and Mayra Chie,

And my brother,

Juan Santi Chong

ACKNOWLEDGMENTS

First, I want to thank my father Santiago Chong, my mother Mayra Chie, and my brother Juan Santi Chong, for the unconditional support and love.

To Dr. Paul Salama and Dr. Paul Territo for all the help and guidance during the development of this project. Without their help and knowledge, the completion of this project could not have been possible.

To Dr. Brian King for his help and support during my master's coursework.

To Dr. Maher Rizkalla for being part of my master's thesis committee.

To Sherrie Tucker for all her help and time since the admission process.

Last, but not least, I want to thank my friends Marcos Ramos and Pedro Hernandez for all the encouragement and support they gave me during my studies.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

$x_{i,j}(t)$	Position of the i-th particle in j dimension at time t
$v_{i,j}(t)$	Velocity of the i-th particle in j dimension at time t
c_1	Cognitive Acceleration Coefficient
c_2	Social Acceleration Coefficient
$p_{best,ij}$	Local best position of particle i for dimension j
$s_{best,j}$	Swarm best position for dimension j
w	Inertia Weight
fps_{ACQ}	Acquisition Frame Rate
N_{size}	Neighborhood Size
I_i	Infrared Frame i
T_{ij}	Affine Transformation Matrix between Frames i and j
P_i	Point Cloud i
F_i^{ij}	Matched Point Cloud i using matches between IR frames i and j

ABBREVIATIONS

API	Application Program Interface
CT	Computed Tomography
DCMTK	DICOM Toolkit
DICOM	Digital Imaging and Communications in Medicine
FLANN	Fast Library for Approximate Nearest Neighbors
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HOG	Histogram of Oriented Gradients
ICP	Iterative Closest Point
IR	Infrared
MRI	Magnetic Resonance Imaging
NNS	Nearest Neighbors Search
OpenCV	Open Source Computer Vision
OpenGL	Open Graphics Library
PET	Positron Emission Tomography
PSO	Particle Swarm Optimization
SDK	Software Development Kit
SIFT	Scale-Invariant Features Transform
SURF	Speeded-Up Robust Features
TOF	Time-of-Flight

ABSTRACT

Author: Chong Chie, Juan Antonio Kim Hoo. MSECE
Institution: Purdue University
Degree Received: May 2017
Title: Motion Correction of PET/CT Images
Major Professors: Paul Salama and Paul Territo

The advances in health care technology help physicians make more accurate diagnoses about the health conditions of their patients. Positron Emission Tomography/Computed Tomography (PET/CT) is one of the many tools currently used to diagnose health and disease in patients. PET/CT explorations are typically used to detect: cancer, heart diseases, disorders in the central nervous system. Since PET/CT studies can take up to 60 minutes or more, it is impossible for patients to remain motionless throughout the scanning process. This movements create motion-related artifacts which alter the quantitative and qualitative results produced by the scanning process. The patient's motion results in image blurring, reduction in the image signal to noise ratio, and reduced image contrast, which could lead to misdiagnoses.

In the literature, software and hardware-based techniques have been studied to implement motion correction over medical files. Techniques based on the use of an external motion tracking system are preferred by researchers because they present a better accuracy. This thesis proposes a motion correction system that uses 3D affine registrations using particle swarm optimization and an off-the-shelf Microsoft Kinect camera to eliminate or reduce errors caused by the patient's motion during a medical imaging study.

1. INTRODUCTION

Advances in health care technologies have helped physicians make more accurate diagnoses about the health and medical conditions of their patients. A consequence of having better diagnosis is that doctors can decide the best plan of action to treat any disease or health related problem. One of the many tools currently used to diagnose health problems in patients is the Positron Emission Tomography-Computed Tomography (PET/CT). PET/CT, is an advanced nuclear imaging technique used to obtain information about the structure and metabolic processes of cells and tissues in the body [1]. PET/CT scans are typically used to detect: cancer, heart diseases, brain disorders and diseases of the central nervous system. In addition, when used to detect cancer, it reveals how the cancer is metabolized, as well as whether it has spread to other parts of the body.

Since PET/CT imaging can take up to 60 minutes or more to acquire, it is likely that patients will not be able to remain motionless throughout the imaging process. Furthermore, for pediatric, geriatric, and neurodegenerative patients, the motion is often involuntary [2] [3] [4]. These movements create motion-related artifacts which alter the quantitative and qualitative results during the scanning process. The patient's motion results in image blurring, reduction in the image signal to noise ratio, and reduced image contrast, which could lead to a misdiagnosis of the patient's medical condition. In some cases, the quality of the images obtained cause the patient to have to be re-imaged, which generates loss of revenue and time, as well as increases the exposure time of the patient to ionizing radiation [5].

As the resolution of the PET/CT scanners increase, the motion correction task becomes increasingly important. In the literature, software and hardware-based techniques have been studied to implement motion correction [6] [7]. From these methods, techniques based on the use of an external motion tracking system have been

developed, and are preferred by researchers because they provide an opportunity to improve accuracy. In addition, marker and marker-less motion tracking systems have been implemented [8] [5] [9].

Of particular interest is a motion tracking system that utilizes Microsoft Kinect. The Microsoft Kinect is a motion sensor device capable of capturing RGB images as well as infrared images and depth maps. It has a tracking library that allows users to develop software capable of capturing and tracking the body and face of a target. But these libraries have limitations with respect to the orientation in which the sensor must be placed, the quantity of joints that can be traced, the position and orientation of the people in the scene, and the number of angles that characterise the movements of the face. For example, Skeletal Tracking which was developed to recognize standing and sitting users in front of the Kinect requires the user to be facing towards the Kinect and their head and upper body must be in the field of view of the sensor in order to be recognized [10]. Since patients would be lying down during a PET/CT scan, if the provided libraries are used, the Microsoft Kinect will not be able to track their movements inside the scanner. Therefore, it is not possible to obtain the necessary information to perform the motion correction.

This thesis proposes a motion correction system that uses 3D affine registrations using particle swarm optimization and off-the-shelf Microsoft Kinect to eliminate or reduce errors caused by the patient's motion during a medical imaging study. The concept is to use the Microsoft Kinect sensor to acquire and store the movements of the patients during the scanning process. Using a series of image frames that are submitted to an algorithm that identifies unique keypoints in the scene. These keypoints are extracted frame by frame and an optimized alignment of the keypoints is constructed using affine registration. The resulting affine transformations between the frames are then used to eliminate the patient's motion in the medical image files.

This thesis is organized as follows: Chapter 1 contains the background and motivation for the proposed motion correction system, while Chapter 2 presents a review of alternative systems that have already been proposed. Chapter 3 details the work-

flow and implementation of the various stages of the proposed system. Chapter 4 includes the results obtained from the implemented system, and Chapter 5 consists of the conclusions obtained and future work that would improve the proposed system.

2. LITERATURE REVIEW

In this chapter, a review of previous researches is presented. This researches are divided in two sections: Section 2.1 presents a series of studies related to motion correction on medical images, and Section 2.2 shows studies related to the use of the Microsoft Kinect and depth information for motion tracking.

2.1 Related Works

There are several research areas related to this project. The use of motion sensing devices for body and head tracking for medical purposes has been studied in [11] [5] [9] [8] and the studies of the effect of motion correction on medical data is shown in [9] [8] [6] [7] [12] [13].

In 2014, the use of the Microsoft Kinect sensor as a tool to monitor safety of a patient during hospital stay was studied [11]. The study proposed a system that can automatically detect edges and posture of the patient's bed using depth information acquire by the Kinect. In addition, the proposed system can work under several different lighting changes without the need to process color information.

Motion due to respiration creates breathing artifacts that affects the quality of the results on a whole body PET scan [5]. These breathing artifacts are a consequence of the acquisition time of PET scan, since it is acquired during several respiratory cycles, which can lead to spatial blurring. This blurring can cause errors such as reduction in the measured uptake, incorrect delineation of the volume, and misalignment with anatomical imaging.

A method to produce a respiratory signal which allows gating of PET list-mode data was developed in [5]. The proposed system uses a Microsoft Kinect sensor to track the entire surface of the chest. This system was compared to an existing

commercial respiratory monitoring system, called "Varian RPM Tracking", which was designed primarily for radiotherapy patients. The results show that the Kinect is able to match the performance of the Varian RPM system in the measurement of the rate of the cyclic motion, and furthermore, the Kinect was more accurate in determining the amplitude of movements.

The importance of motion correction for brain PET images was studied in [9]. In this study, the authors presented a system that uses the Kinect as a marker-less tracking device. This system was compared to an existing marker-based tracking system, called Polaris Vicra. A drawback of marker-based tracking systems is that the method for attaching the markers to a patient's head may be uncomfortable and may move during the scanning, which may lead to unreliable tracking information. The results of this study demonstrated that Polaris Vicra has higher noise levels than the Kinect. In addition, the precision, robustness and stability of the Kinect based system were tested, and the results proved that the Kinect can be used as a motion correction scheme for high resolution brain PET.

An alternative approach for a 3D marker-less tracking system for motion correction was developed in [8]. The proposed system in this research, called Tracoline, consists of two Point Grey Research Inc CCD cameras and one Texas Instruments PICO projector modified to operate in infrared, thus avoiding discomfort to the patients. This system is capable of correcting motion in PET brain images without compromising accuracy. The performance of this system was compared with Polaris Vicra. Three experiments were performed: one using a phantom scan and two using human scans. During the experiments, both systems were used simultaneously and the results obtained to the PET images with motion correction were compared with the PET images without motion correction. In the experiment using the phantom, the tracking conditions were optimal for both tracking systems and no differences were noticed. Similarly, during the experiments using human scans, both systems

exhibited similar accuracies and the only difference observed was that the marker-less system eliminated errors due to the attachment of the markers to the patient and improved the overall work-flow.

A survey of the performance and accuracy of three motion compensation methods was performed in [6]. The three studied methods are: frame-based motion correction, post-reconstruction image registration, and event-by-event motion compensation with list-mode reconstruction, also known as MOLAR. The results obtained in this study show that event-by-event motion compensation with list mode has a superior motion correction than the other two methods, especially if motion is larger than 10 mm. Also, event-by-event motion compensation with list mode can reliably correct all reasonable head motions.

A comparison between event-by-event motion correction and frame-based motion correction in human brain PET imaging was described in [7]. It was concluded that given enough and accurate motion information, event-by-event motion correction is able to reliably correct head motion. On the other hand, frame-based motion correction was able to give comparable quantitative accuracy if used correctly with an aligned attenuation map and externally acquired motion data. In addition, frame-based motion correction is reliable and accurate when the motion is less than 5 mm and the attenuation map is correctly aligned, otherwise, given large motion, event-by-event motion correction was preferred.

A study of the uses of retrospective motion correction in the acquisition of high-quality high-resolution anatomical magnetic resonance imaging (MRI) images can be found in [12]. Even though they consider that the results are potentially biased, the average retrospective motion correction distance was 0.56 mm and the corrected images demostrated a significant improvement over the images without correction. The conclusion at the end of this study was that the proposed technique can be used to reliably improve the quality of the images in high-resolution MRI images, and suggests that shortening the acquisition times could lead to further improvements in the quality of the motion correction.

2.2 Kinect and Depth Data used for Tracking

The Microsoft Kinect has been widely used as a tool for motion tracking, face detection, and head pose estimation [14] [15] [16]. In addition, research studies have been conducted to assess the accuracy of the depth information obtained from the Microsoft Kinect [17]. Lastly, Microsoft developed a tool named KinectFusion, which provides 3D object scanning and 3D model creation using a Microsoft Kinect Sensor [18] [19].

Khoshelham [17] evaluated the quality, accuracy and density of the depth information provided by the Kinect and concluded the following: first, if the Kinect is properly calibrated, the point cloud generated by the depth data does not have significant systematic errors compared with a laser scanning data. Second, as the distance from the sensor increases, the error of the depth measurements increases quadratically until reaching a maximum of 4 cm at the maximum range. Third, as the distance to the sensor increases, the density of points decreases. The density of points is influenced by the depth resolution, which is small at large distances. Moreover, Khoshelham recommended at the end of the study that the information be acquired at a distance of 1 to 3 meters from the sensor since, at large distances, the quality and accuracy of the depth data becomes degraded by the effect of the noise and the lower resolution of the depth measurements [17].

A tracking system that uses depth edges was described in [16]. This work concluded the following: the system is invariant to the lighting conditions as long as the environment does not have a light source, the error in the depth measurements is approximately 5 to 15 mm for a distance of 0.5 to 3 m, and for some objects, such as light absorbing materials, it is not possible to obtain a distance.

The first KinectFusion research article [18] introduces the concepts behind KinectFusion, an interactive 3D reconstruction system. KinectFusion takes real-time depth information from a moving Kinect and creates a single accurate high quality 3D model of the scene. The algorithm is able to create a 3D model of an indoor scene within

seconds. The major contribution of this research is the creation of a detailed GPU pipeline that is capable of achieving tracking, reconstruction, segmentation, rendering and interaction, in real-time and in 3D.

The second KinectFusion research article [19] expands the previous research [18] by creating an accurate real-time system capable of mapping complex and arbitrary indoor scenes under varying lighting conditions using the Microsoft Kinect. The system uses all the information acquired from the camera to generate a surface model of the observed scene. In addition, the proposed system works at a frame rate of 30 Hz and is invariant to the scene's lighting conditions because it only uses the depth data. Also, this research compares the advantages of tracking against a growing full surface model over frame-to-frame tracking. With this research, the creators believe that the proposed system will become a facilitator for augmented reality and interaction scenarios, in addition to providing a high quality occlusion handling.

3. APPROACH

This chapter presents the implementation of the project. It contains the algorithms used for the development of the proposed motion correction system, as well as the selection of the parameters used in these algorithms.

3.1 Overview

The proposed system aims to correct motion artifacts in PET/CT images generated by the motion of the patients inside a scanner. The system uses a Microsoft Kinect to obtain infrared and depth images over the duration of the PET/CT scan, that are used to correct the medical image files obtained from the PET/CT scanner. All image information required by the system for motion correction, such as the image slice, its location, and the time at which it was obtained; are contained in the medical image files generated by the PET/CT scanner. The Digital Imaging and Communications in Medicine (DICOM) standard is the standard used in medical imaging to handle, store, print and transmit information acquired by medical devices.

For the development of the project, the following libraries are used:

1. Kinect SDK

   Kinect SDK is a software development kit that allows users the ability to develop applications using a Microsoft Kinect as an input device.

2. Open Source Computer Vision

   Open Source Computer Vision (OpenCV) is a library of computationally efficient functions for real-time computer vision applications. In this project, OpenCV libraries are used to extract and match features of the infrared images obtained from the Microsoft Kinect.

3. Open Graphics Library

   Open Graphics Library (OpenGL) is an application programming interface used for rendering 2D and 3D vector graphics. In this project, OpenGL is utilized to handle the creation, manipulation and visualization of the point clouds using the depth maps obtained from the Microsoft Kinect.

4. DICOM Toolkit

   The DICOM Toolkit (DCMTK) is a package that contains a set of libraries and applications whose purpose is to implement part of the DICOM standard. The toolkit was is used in this project for the manipulation of the DICOM files obtained from the PET/CT scanner.

Motion correction is performed by obtaining an affine registration between two point clouds of a user defined region of interest in the scene. Two techniques are used for this: SURF and PSO. Subsequently, the result of the affine registration step (which is a transformation matrix between the two point clouds) is applied to the PET/CT images. In order to choose the corresponding transformation matrix, a time stamp alignment process is used.

In of this project, it is assumed that the movements of the patient inside the PET/CT scanner can be approximated by affine rigid transformations. This stems from the assumption that between consecutive frames little or no movement is observed. Furthermore, in case there is no apparent movement observed between consecutive frames, the result of the registration will be an identity matrix. In addition, only the regions selected by the user will be corrected. An important consideration for this project is that the system must be able to work regardless of the lighting conditions in the room, since it cannot be guaranteed that the lightning conditions will remain the same during a PET/CT scan. Also, the system does not work in real-time and uses a marker-less tracking system. Motion correction is performed in the image domain and only once the scan is complete.

3.2 Background

In this section, a review of all the necessary tools for the development of this research is presented. The rest of this section is organized as follows: Section 3.2.1 describes Image Registration, Section 3.2.2 Affine Transformations, Section 3.2.3 Affine Registration, Section 3.2.4 Non-Linear Registration, Section 3.2.5 Features Detection, Extraction and Matching, Section 3.2.6 Particle Swarm Optimization Background, and Section 3.2.7 Microsoft Kinect.

3.2.1 Image Registration

Image registration refers to a method wherein a transformation is applied to an image, a series of images, or a video [20]. In general the transformation maps the pixels of an input image into corresponding pixels on a second image, usually referred to as the reference image. The most common examples of transformations applied to images are: geometric, rigid, affine, projective, and perspective transformations. Estimating the transformation parameters allows the comparison of images from different point of views or different time frames. In addition, these parameters can be used to determine motion, depth, or distances between a pair of images.

Image registration can also be used to align two or more images into one common reference. The images can be taken from different viewpoints, different times or using different sensors. Usually, the correspondences between the images are unknown *a-priori*. The image that is kept unchanged and used as the common reference is called the "reference frame", while the other images "registered" to the reference frame are called the "input frames". The purpose of image registration is to estimate the "optimal" transformation such that the transformed input frames become similar to the reference frame.

Image registration algorithms can be divided into two categories: rigid and non-rigid registration. Rigid registration methods utilize rigid transformations that do not change the distances between points [21]. On the other hand, non-rigid registration schemes use non-rigid transformations that employ non-linear transformations that include articulated objects or objects that change shape over time [22].

Both, rigid and non-rigid registration techniques are used in robotics, augmented reality, and health-care. The shared goal between these three different fields is to determine the position and/or orientation of an object given a reference viewpoint. Registration algorithms are not limited to 2D images, they can be applied to 3D objects in applications such as 3D scanning (e.g. Medical Imaging), 3D localization, and human body and faces detection. Image registration applications can be divided into four major groups based on the acquired input images [23]:

- Multiview analysis: The images are acquired from different viewpoints. The goal is to gain a larger 2D view of the scene or a 3D representation of the scene. An example of this application is image mosaicing.

- Multitemporal analysis: The images are acquired at different times. The goal is to find changes in the scene during a period of time. An example of this application is motion tracking.

- Multimodal analysis: The images are acquired from different cameras or sensors. The goal is to gather information from different sources to obtain a detailed representation of the scene. An example of this application is the employment of various sensors to record the anatomical body structure as well as functional and metabolic body activities.

- Scene to model registration: The acquired images from a scene are registered to a model of the scene. The goal is to find an acquired image in the represented model. An example of this application is automatic quality inspection.

3.2.2 Affine Transformations

An affine transformation is a linear transformation that preserves points, straight lines, parallel lines and planes, but does not necessarily preserves angles and distances. However, the distance ratios between points on a straight line are preserved.

An affine transformation can be expressed as a matrix vector product. The matrix includes all the affine transformations parameters, while the vector includes the pixel coordinates of the input object.

In the case of 2 dimensions this product is expressed as [21]:

$$p_{i,2D} = \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & T_x \\ C & D & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = T_{M,2D} \times q_{i,2D} \tag{3.1}$$

where, $q_{i,2D}$ is the the i^{th} element of the input object, $T_{M,2D}$ the transformation matrix, and $p_{i,2D}$ the i^{th} element of the transformed object. Equivalently, for the 3D objects, the transformation can be expressed as [21]:

$$P_{i,3D} = \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C & T_x \\ D & E & F & T_y \\ G & H & I & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = T_{M,3D} \times Q_{3D} \tag{3.2}$$

where, $Q_{i,3D}$ is the the i^{th} element of the input object, $T_{M,3D}$ the transformation matrix, and $P_{i,3D}$ the i^{th} element of the transformed object.

Affine transformations include translations, scaling, reflection, rotation, shearing and/or any combination thereof. These transformations are explained in the following sections.

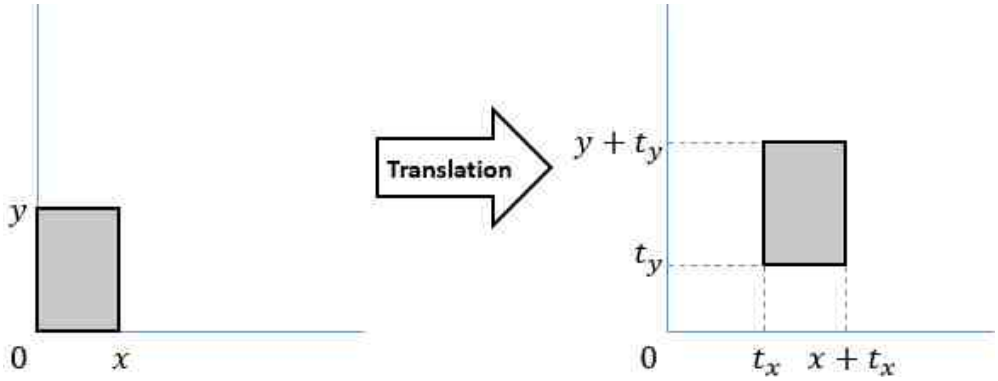


Figure 3.1. Example of a translation in 2D

Translation

Translation is a linear transformation that moves every point of an object by the same amount in a given direction. The matrix representation for a translation in the 2D case is [21]:

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

and in the 3D case it is [21]:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Figure 3.1 shows an example of a translation in 2D space.

Rotation

Rotation is a linear transformation that rotates an object by a specified angle θ in the clockwise direction around a reference point, called center of rotation. In general, the rotation matrix in 2D, around the origin, is [21]:

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

For the 3D case, there exists three cases for rotation, one around each axis in the 3D Cartesian coordinate system. These matrices are [21]:

- Rotation around the x-axis,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

- Rotation around the y-axis,

$$\begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

- Rotation around the z-axis,

$$\begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

Figure 3.2 shows an example of a rotation about the origin on the 2D space.

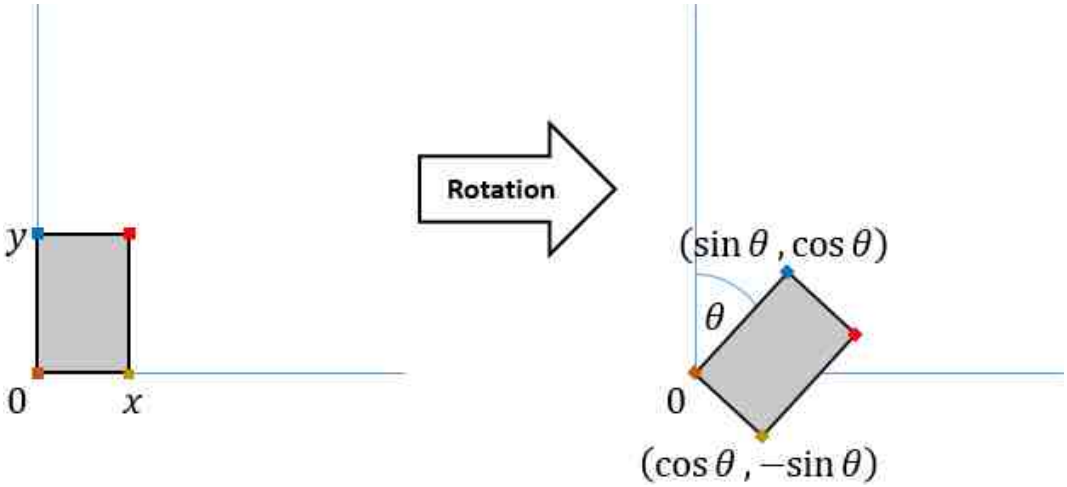


Figure 3.2. Example of a rotation about the origin in 2D.

Reflection

Reflection is a linear transformation that reflects an object relative to a reference line, called the line of reflection. For the 2D case, there are three cases of reflection [21]:

- Reflection about the origin (see Figure 2.3)

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

- Reflection about the x-axis (see Figure 2.4)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

- Reflection about the y-axis (see Figure 2.5)

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Figures 3.3, 3.4, and 3.5 depicts examples of reflections about the origin, $x-$axis and the $y-$axis, respectively.

For the 3D case, reflections are performed about planes [21].

- Reflection about the xy-plane,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

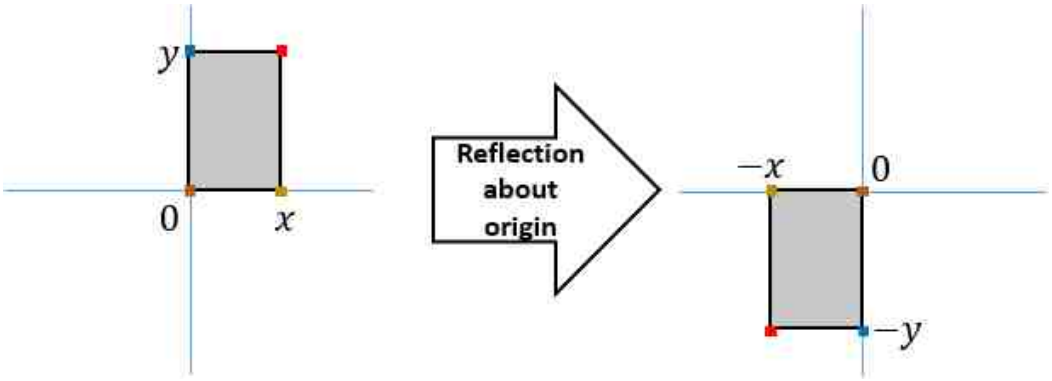


Figure 3.3. Example of a reflection about the origin in 2D

- Reflection about the yz-plane,

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

- Reflection about the xz-axis,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

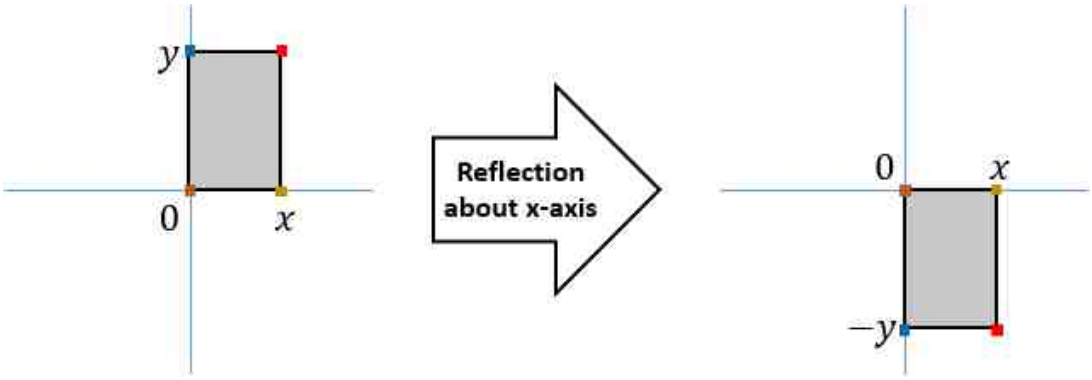


Figure 3.4. Example of a reflection about the x-axis in 2D

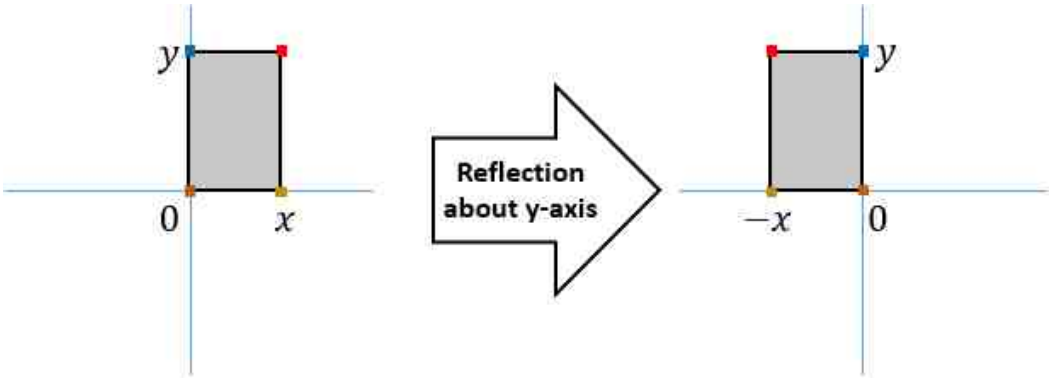


Figure 3.5. Example of a reflection about the y-axis in 2D

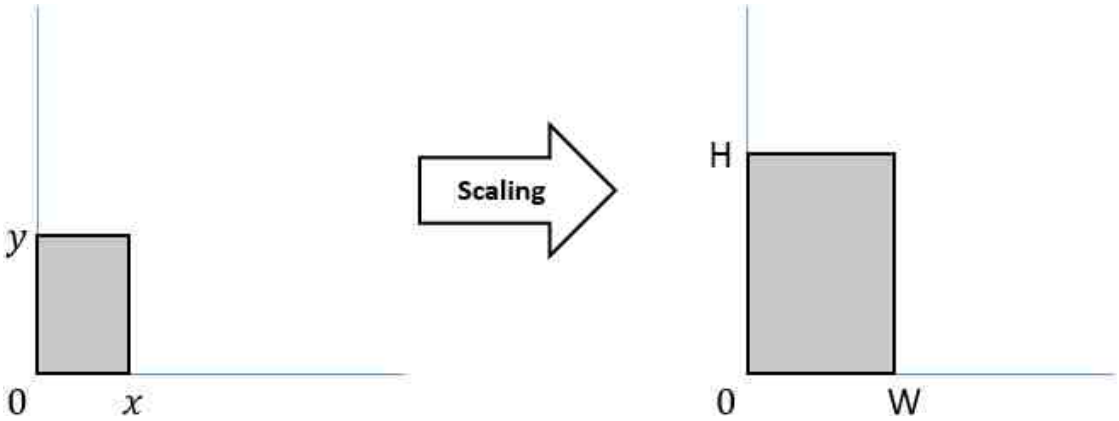


Figure 3.6. Example of a scaling about the origin in 2D

Scaling

Scaling is a linear transformation that enlarges or shrinks an object by a scale factor. When the scale factor is the same for all directions, the scaling is considered uniform scaling. When at least one of the scaling factors is different than the others, it is referred to as non-uniform scaling. In general, the scaling matrix in 2D is [21]:

$$\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

and in 3D is [21]:

$$\begin{bmatrix} W & 0 & 0 & 0 \\ 0 & H & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

Figure 3.6 shows an example of a scale transformation in 2D.

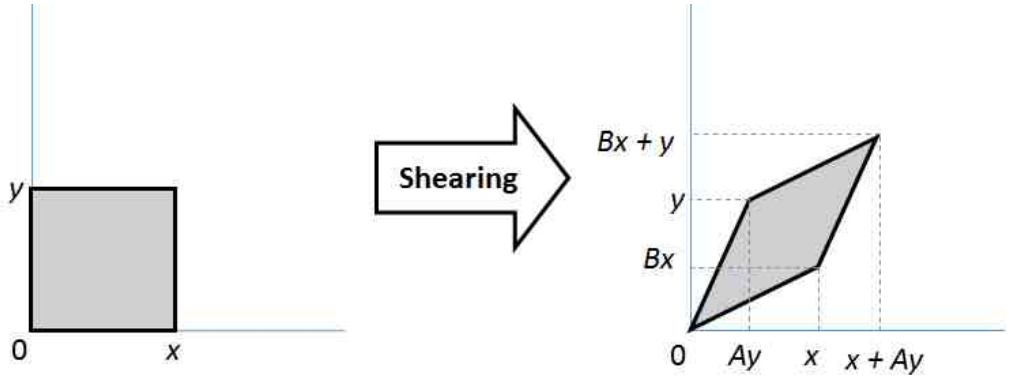


Figure 3.7. Example of shearing in both axis.

Shear

Shearing is a linear transformation that displaces each point in a fixed direction. In general, the shearing matrix in 2D is [21]:

$$\begin{bmatrix} 1 & \mathrm{A} & 0 \\ \mathrm{B} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

and in 3D is [21]:

$$\begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

Figure 3.7 shows an example of a shearing transformation.

3.2.3 Affine Registration

Affine registration is the process of obtaining the corresponding affine transformation between two objects. The number of parameters that can vary independently without violating imposed constraints on a system are called degrees of freedom. In the 2D case, the transformation consist of 6 degrees of freedom, whereas, in 3D it has

up to 12 degrees of freedom. Using either Equation 3.1 or Equation 3.2, solving for an affine registration can be reduced to the operation of finding the corresponding transformation matrix ($T_{M,2D}$ for the 2D case, or $T_{M,3D}$ for the 3D case), given two objects (p_{2D} and q_{2D} for the 2D case, or P_{3D} and Q_{3D} for the 3D case).

3.2.4 Non-linear Registration

In addition to linear transformations, such as affine transformations, objects can also suffer from deformations. A transformation which allows non-uniform mapping between objects, is called a Non-Linear Transformation. The degrees of freedom for a non-linear transformation can vary from 7 to 1000, generating a more complex transformation. These types of transformations are common in medical image registration since events such as, breathing and anatomical changes can produce an elastic transformation of the data of the same patient obtained at different times.

Some examples of non-linear transformations include:

- Thin plate spline.

  This is a spline-based technique introduced by Duchon [24], which is used for data interpolation and smoothing. It is used as a non-rigid model for image alignment and shape matching. The popularity of thin plate spline is due to some of its properties, which include the production of smooth surfaces, no need for manual tuning, and the fact that it has a closed solution for warping and parameter estimation.

- Cubic-spline

  This is a spline-based technique in which the spline is constructed using a third-order polynomial, which has to pass through a set of control points [25]. The advantages of using cubic-splines are, simplicity of construction, ease and accuracy of evaluation, and the capacity to approximate complex shapes using curve fitting and design.

3.2.5 Features Detection, Extraction and Matching

A feature, also called a keypoint, is a point of interest used to describe an object or region in an image. Normally, a feature is represented by its spatial location in the image. Features are important because no matter which transformation is used on an image, it should be possible to obtain the same feature in the transformed image. In image processing, a feature could be a structure such as a point, an edge or an object in the image. Usually, features are obtained using a general neighborhood operation or a feature detection algorithm. In this section feature detection and extraction algorithms, such as: SIFT and SURF, and feature matching algorithms will be discussed.

Features Detection and Extraction

Feature detection methods are used to find key-points in an image. Feature detection can be classified via two types of algorithms: feature detector and feature descriptor. A feature detector takes an image as input and returns as an output spatial locations of interesting regions in the image. A corner detector is an example of a feature detector, since it returns the locations of corners in the input images but it does not returns any additional information about the detected features. On the other hand, a feature descriptor takes an image as input and returns features descriptors, also called features vectors.

A feature descriptor is a data vector that describes the local structure around a feature. Features descriptors are used to differentiate one feature from another. Ideally, these descriptors are invariant under any transformation applied to the image.

After detecting the features, it is necessary to obtain an efficient descriptor that describes the area around each obtained feature. Feature extraction methods are used to obtain the descriptors from each pixel around the feature. Feature extraction refers to a type of dimensionality reduction used to represent regions of interest in an image

in a compact and efficient way. Some examples of features descriptors and features extraction algorithms are: Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF) and Histogram of Oriented Gradients (HOG).

SURF and SIFT are two feature detection/extraction algorithms that are scale and rotation invariant. The difference between them is that SIFT is better at handling changes in viewpoints and illumination while SURF is better at handling blurs and rotations [26]. In addition, the developers of SURF claim that SURF is faster than SIFT [27].

Scale-Invariant Feature Transform

Scale-Invariant Feature Transform (SIFT) was first introduced by Lowe in 1999 [28], where it was is used to detect and describe local keypoints in images. The features and descriptors generated by SIFT are used in problems, such as point matching between different views, object tracking, and view-based object recognition. SIFT is invariant to translations, rotations, and scaling in the image domain, while being robust to perspective and illumination changes [28].

The SIFT algorithm can be split into the following steps:

1. Construct a scale-space of images.

   To create the SIFT scale-space of images, start by taking the original image and construct a set of progressively Gaussian blurred images. This group of images is called an octave. Then, generate a new octave by resizing the original image using a factor of 0.5, and generate a new set of progressively Gaussian blurred images. This last step is repeated several times. Lowe [28] suggests that the ideal number of octaves and blur levels for the algorithm is 4 and 5, respectively.

2. Laplacian of Gaussian Approximation.

   The Laplacian of an image is an operation used to highlights the areas of rapid intensity change in an image. It is typically used for edge and corner detection. In general, the Laplacian of an image is defined as [29]:

$$L(x, y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{3.19}$$

   The Laplacian of Gaussian (LoG) is the process of using a Gaussian blur before applying the Laplacian operator. The two major concerns of the Laplacian are that it is extremely sensitive to noise, and calculating second order derivatives are computationally intensive operations (because these types of operations require large amount of memory resources and elevated computational time). Therefore, the SIFT algorithm uses an approximation of the Laplacian of Gaussian called: Difference of Gaussians (DoG).

   The Difference of Gaussians is obtained by taking the difference between two consecutive images within a octave. This process is repeated with all the consecutive pairs in an octave, and is performed for all octaves. The resulting difference images are approximately equivalent to the results that can be obtained by the Laplacian of Gaussian. The two benefits obtained by using Difference of Gaussians are that the process of computing second order derivatives is replaced with simple subtraction, and the images generated by the Difference of Gaussians are scale-invariant.

3. Determine the Keypoints.

   The process of finding the keypoints consists of two parts:

   (a) Locate maxima and minima in the Difference of Gaussians images.

   To locate the maxima and minima points, iterate through each pixel and check all its neighbors. The comparisons are made using eight neighbors in the current image and nine neighbors in the scale above and below. If the pixel is the largest or smallest of all its 26 neighbors, then the

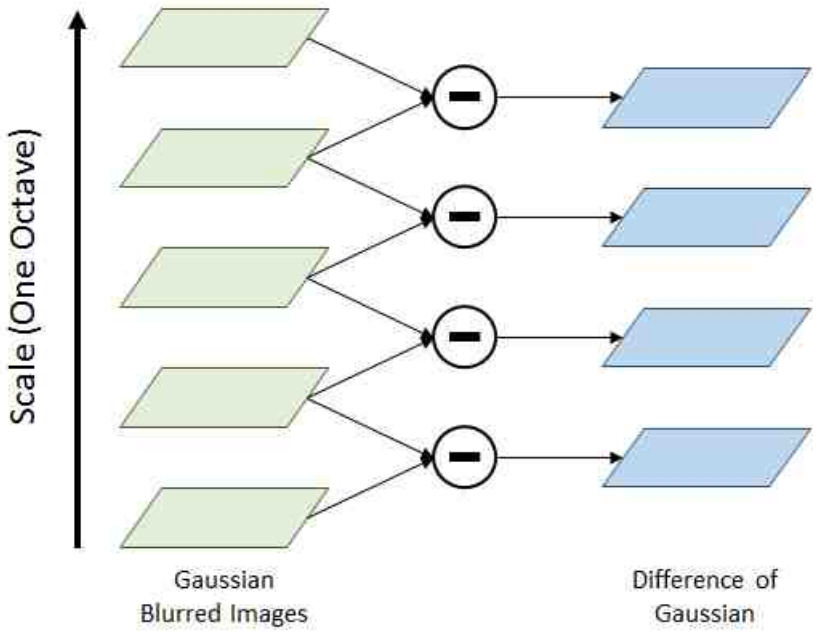


Figure 3.8. Difference of Gaussian

pixel is marked as an approximate maxima/minima. These are considered approximate because, typically maxima/minima point do not lie exactly on a pixel, but somewhere within the pixel.

(b) Find subpixel maxima and minima.

Since subpixel locations are not of maxima/minima are not physically available thus must be obtained mathematically. This is done using a quadratic Taylor series expansion of the Difference of Gaussians image around the approximate maxima/minima location. Obtaining the Taylor series expansion (See Equation 3.20) will provide the subpixel location of the keypoint,

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{3.20}$$

here the Difference of Gaussians image D and its derivative are evaluated at the keypoint $\mathbf{x} = (x, y, \sigma)^T$.

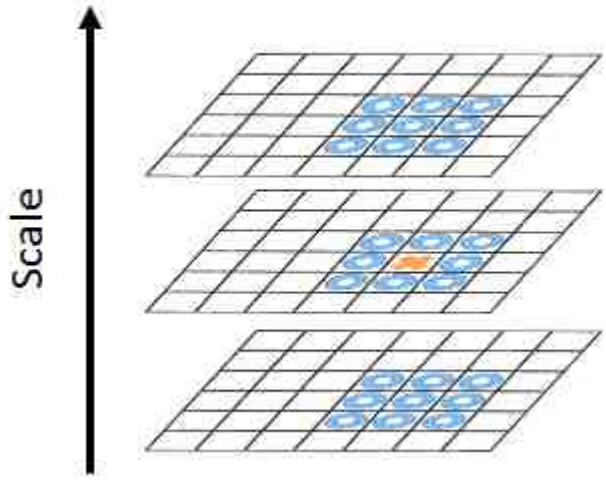


Figure 3.9. Marked Keypoint

Lowe [28] recommends the use of 2 extrema images per octave. The number of Difference of Gaussians images needed to generate 2 extrema images is 4. To generate 4 Difference of Gaussians images, requires 5 Gaussian blurred images, therefore, the recommended number of blur levels is 5 in each octave.

4. Eliminate low contrast keypoints.

   The previous steps generates a large amount of keypoints, but some of them have low contrast or lie along edges, and are not useful as features. Removing low contrast keypoints requires one comparison: if the magnitude of the absolute intensity at the current pixel in the Difference of Gaussian image is less than a threshold, the keypoint is removed.

   Removing edges requires the calculation of two gradients at a keypoint, which are orthogonal to each other. Based on the area around the keypoint, there exists three possible cases:

   (a) Flat region: Both gradients will have small values.

   (b) Edge: One gradient will have a large value, which means it is perpendicular to the edge, and the other will have a small value, which means it is along the edge.

   (c) Corner: Both gradients will have large values.

   From these three cases, corners represent important keypoints. Therefore, if the values of both gradients are above a threshold, the keypoint is kept, otherwise, it is removed. The mathematical tool used to achieve this task is the Hessian Matrix, which permits checking if a point is a corner or not. SIFT uses a criterion based on the ratio between the eigenvalues of the Hessian Matrix.

5. Assign an orientation to each keypoints.

   To provide rotation invariance, an orientation must be assigned to each keypoint obtained from the previous step. In this case, the directions and magnitudes of the gradients around each keypoint must be collected, then, the most dominant

orientation in the keypoint region is assigned as the orientation for that keypoint. To ensure rotation invariance, all the calculations in the following steps for the keypoint are done relative to the assigned orientation.

The gradient magnitude and orientation are given by Equations 3.21 and 3.22:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (3.21)$$

$$\theta(x,y) = \tan^{-1} \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \quad (3.22)$$

where $L(x,y)$ is the Laplacian of Gaussian approximation for the pixel located at position $< x, y >$.

The gradient magnitude and orientation are obtained for all the pixels in the orientation collection region. The size of the orientation collection region is equal to the size of the Gaussian blur kernel. An orientation histogram is also created using the collected gradient orientations. This histogram is composed of 36 bins of 10 degrees each which covers the 360 degrees of orientation. Each sample pixel is added to its corresponding bin with a value proportional to its gradient magnitude.

The peak of the orientation histogram corresponds to the dominant orientation, and this orientation is assigned to the given keypoint. Also, if the histogram has any peaks above 80% of the highest peak, each of these peaks is used to create a new keypoint with the same location and scale but with an orientation equal to that of the new peak.

6. Generate the SIFT Features/Descriptors.

   After keypoints have been identified, a SIFT feature structure is obtained at each keypoint in the scale space. The feature structure is composed of four elements: location of the keypoint in the image p, scale s, orientation r, and a 128 dimensional descriptor vector f generated from the local image gradients. The SIFT feature structure is denoted by:

$$\text{SIFT Feature} = < p, s, r, \vec{f} > \quad (3.23)$$

To generate the 128 dimensional vector, a 16x16 window is placed around each keypoint. The 16x16 window is divided into sixteen 4x4 windows. For each window, the gradient magnitudes and orientations is obtained, which will generate an eight bin orientation histogram. To accomplish rotation independence, the keypoint rotation is subtracted from each orientation. As in the previous orientation assignment step, the value added to the bin is proportional to the gradient magnitude, but it also depends on the distance from the keypoint. This will reduce the pixels in the 4x4 window into eight predetermined bins. This is done for all sixteen windows. At the end, the 16X16 window will generate 128 numbers: 16 windows times the 8 orientation bins. After normalizing these 128 values, the result is the descriptor vector, which uniquely identifies a keypoint.

Speeded-Up Robust Features

The two main concerns of SIFT are its execution time and susceptibility to illumination changes. An alternative features detection and extraction algorithm developed to overcome this issues is Speeded-Up Robust Features. Speeded-Up Robust Features (SURF) was introduced by Bay, Tuytelaars and Van Gool in 2006 [30]. It is a feature detector and descriptor algorithm. As with SIFT, the algorithm is used to address problems such as object recognition, object tracking, image registration and 3D reconstruction. SURF is partly inspired by the principles of SIFT.

The first difference between SIFT and SURF is the Laplacian of Gaussian approximation. While SIFT uses Difference of Gaussians, SURF approximates the Laplacian of Gaussian with a Box Filter. The advantage of this approximation is an increase in speed, because convolution with a box filter is quickly obtained using integral images. Integral images, also known as Summed Area Table, is technique used to quickly and efficiently calculate the sum of pixel values in a given image [31]. In addition,

this process can be performed in parallel for different scales. For scale and location, SURF uses the determinant of the Hessian Matrix, as a measure of change around a keypoint, and the points where the determinant is maximum are marked as keypoints.

The orientation assignment in SURF is performed using wavelet responses in the horizontal and vertical directions using a circular neighborhood size of $6s$, where s is the scale at which the point is being detected. The obtained wavelet responses are then weighted by a Gaussian function centered at the keypoint. The dominant orientation is obtained by obtaining the sum of all wavelet responses within a sliding orientation window of size 60 degrees. The sum of the horizontal and vertical responses within the window yield an orientation vector, and the longest obtained vector defines the orientation of the keypoint.

Wavelet responses are also used by SURF to obtain the feature descriptors. A square region of size $20s$x$20s$ centered at the keypoint is extracted, where s is the scale in which the keypoint was detected. This region is divided into 4x4 sub-regions, and for each sub-region, horizontal and vertical wavelet responses are found. Each sub-region generates a vector composed of the wavelet responses. The concatenation of these vectors results in a 64 dimensional SURF feature descriptor. The SURF feature descriptor can be extended to a 128 dimensional version which leads to slower computation but a slight increase in accuracy. To extend the vector, the sums of the horizontal wavelet responses are found separately depending on the sign of the vertical wavelet response. In the same way, the sums of the vertical wavelet responses are obtained separately depending on the sign of the horizontal wavelet response.

In this research the algorithm used for feature detection and extraction is SURF because SURF has a faster runtime and it is more robust to changes in illumination.

Feature Matching

Feature matching refers to the process of finding correspondences between features from two similar data sets. These correspondences can be done, for example, based on the search distances between features, the euclidean distances between features or differences in the descriptors of the features. Some examples of features matching algorithms include: brute-force matching, iterative closest point (ICP), and nearest neighbors algorithm.

Brute-Force Matching

Brute-force matching works by taking one feature from the first object and comparing it to all the features in the second object using "distance" metric such as Euclidean distance. Then, the closest feature on the second object is returned as the matched feature. This is repeated for each feature in the first object.

Iterative Closest Point

In order to match objects it is necessary to minimize the difference betweem them. One method that can be used to achieve this is the Iterative Closest Point (ICP) method which has been used for such applications as reconstruction of 2D and 3D surfaces, and object registration, and in robotics, for the optimum planning of a robot's route. ICP works by iteratively reviewing the transformation created by a combination of translations and rotations needed to minimize the distance between the two objects. The steps involved are:

1. Define the source object, the reference object, and the stopping criteria.

2. For each point in the source object, find its "closest" point in the reference object.

3. Estimate the transformation using an error cost function which will best align each point in the source object to its corresponding match found in the previous step.

4. Apply the obtained transformation to the source object.

5. Iterate from step 2 until the stopping criteria is met.

Nearest Neighbors Search

As mentioned above, object matching requires finding "closest" points. A technique for achieving this is Nearest Neighbor Search. Nearest Neighbor Search, also referred as proximity search, is an optimization problem that find the closest point in a set of points to a given point, where "closeness" is expressed using a function. Typically, a dissimilarity function is used, for which less similar objects have greater values. Nearest neighbor search appears in several fields, including pattern recognition, robotic sensing, statistical classification, computer vision, and computational geometry.

In the literature, there exists several variants of nearest neighbors search:

- k-Nearest Neighbor [32]

  The k-Nearest Neighbor method searches for the "top" k-nearest neighbors of a given point. The typical use for this technique is to estimate or classify a given point based on its neighbors.

- Approximate Nearest Neighbor [33]

  This technique returns a "good guess" of the nearest neighbor of a given point. Even though it does not guarantee to return the best nearest neighbor, this technique has an improved speed and memory usage.

- Nearest Neighbor Distance Ratio [34]

  Nearest Neighbor Distance Ratio compares the distances between a given point

and its two second closest neighbors. Then, if the distance ratio is below a threshold, the technique validates the match given by the closest neighbor. This technique can be used to recover pictures in content-based image retrieval and for matching problems.

- Fixed-Radius Near Neighbors [35]
  Fixed-Radius Near Neighbors efficiently finds all the neighboring points in the Euclidean space, within a given fixed distance, of a given point.

- All Nearest Neighbors [36]
  The All Nearest Neighbors method attempts to find the nearest neighbors of a group of N data points simultaneously. One way to solve this problem is to apply the nearest neighbor search to each of the N data points. Alternatively, the All Nearest Neighbors technique exploits any redundancy between the N queried points, to obtain a more efficient search. A simple example of redundancy between the queried points is the fact that the "distance" between point X and point Y is the same as the distance between point Y and point X, which means that the same computation can be reused.

k-d Tree

Finding "closests" points requires that these points be organized in such a way to facilitate the search and computations. A method for organizing points is the k-d Tree data structure that was developed by Jon Louis Bentley in 1975 to divide and organize points in a k-dimensional space [37]. One important application for the k-d trees is searches using multidimensional search keys (e.g. nearest neighbor searches). A k-d tree is a special case of the binary space partitioning (BSD) tree, because k-d trees use only the planes perpendicular to the axes of the coordinate system, whereas the planes used in BSD trees can be arbitrary. All nodes of the k-d trees store a point in the k-dimensional space thus each plane must pass through one of the points

in the tree. There are several ways to choose the planes, therefore there are several ways to generate the k-d tree for a set of points. Usually, the following two conditions are used:

- The method used to generate the tree will cycle through the axes used to select the planes, as it descends through the tree. For example, using three axes (x, y, and z), the root of a tree uses a plane aligned to the x-axis, its children will use planes aligned to the y-axis, its grandchildren will use planes aligned to the z-axis, its great-grandchildren will use planes aligned with the x-axis, and so on.

- At each step, the point selected to create and choose the cutting plane will be the median of all the points placed in the k-d subtree.

This method generates a balanced k-d tree, where each leaf node has the same distance to the root node. A general algorithm used to create a k-d tree is the following:

k**dTree**(*points*, *depth*)

1. If the number of points, 'n', is equal to 0, return null.
2. Select the *axis* based on the *depth*. [Use the axis number *depth* mod k].
3. Sort the *points* using the selected *axis*.
4. Choose the *median* of the sorted *points*.
5. Create the *node* and use the *median* as its location.
6. Construct the left subtree using all the points before the median and increase the depth by 1. [k**dTree**(points[0,median-1], depth+1)]
7. Construct the right subtree using all the points after the median and increase the depth by 1. [k**dTree**(points[median+1,n], depth+1)]
8. Return *node*.

3.2.6 Particle Swarm Optimization Background

Finding the "optimal" transformation parameters needed to perform image registration requires the use of an optimization method. In this research, the optimization scheme used is Particle Swarm Optimization. Particle Swarm Optimization is a stochastic computational technique introduced by James Kennedy and Russell C. Eberhart in 1995 [38]. It is a powerful population-based search algorithm that is used to efficiently optimize a wide range of functions in a multi-dimensional space. The roots of Particle Swarm Optimization (PSO) reside in swarm behavior and social interaction from many species in nature. In particular, it was inspired by the behavior and movement of bird flocks and fish schools.

In general, PSO guides a population, called swarm, through a multi-dimensional solution space until a potentially "optimal" solution is reached. Each member of the swarm, called a particle, represents a candidate solution to the problem and the success of each particle influences the action of the swarm.

The original idea used to develop and explain PSO is as follows: consider a flock of birds searching for food in a region. At the beginning, before the birds start to search for food, they can be either scattered around the potential search space or grouped together. During the process of searching for food, one bird can find food better than others in the cohort, which means that this bird is able to notice the promising area where the food may be found, and therefore it has the better information. Then, this bird shares the obtained information with the other members in the flock. Since the flock is communicating, the birds will eventually converge to the area where the food can be found.

From the Particle Swarm Optimization perspective, the bird flock represents the solution swarm; the area in which the bird flock is searching for food represents the solution space; the movements of each bird represents the progress made by the solution swarm in finding the answer; the information represents the most positive solution during the exploration of the solution swarm in the solution space; and the

place where the bird flock finds the food represents the solution of the problem. Particle Swarm Optimization obtains the solution by the joint action of the particles in the swarm, where each particle is allowed to have a regulated behavioral pattern used to show the complexity of the whole swarm. Using this approach, Particle Swarm Optimization can be used to solve complex deterministic problems.

Basic Particle Swarm Optimization Algorithm

A basic particle swarm optimization algorithm contains a population, called a swarm of particles, where each individual particle represents a candidate solution for the problem. These particles then "fly" through a multidimensional search space (i.e. solution space), where the position of each particle is updated based on its own experience and the experience of its neighbors. The changes in a particle's position take into consideration three elements: the particle's inertia, the particle's individual experience, and the swarm's experience. The position of the particle changes through each iteration by adding a velocity term to the current position. This is described as [39]:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3.24}$$

where $x_i(t)$ is the position of particle i in the solution space at time step t, and $v_i(t)$ is the velocity term of particle i at time step t.

The velocity term $v_i(t)$ drives the optimization process, and reflects the experience of the particle and the exchanged information from the particle's neighborhood. The particle's experience is normally referred as the cognitive component, and it depends on the distance between the particle's current position and its own best result since the first iteration. The exchanged information is normally referred to as the social component, and it depends on the distance between the particle's current position and the neighborhood's best result since the first iteration. Based on the size of the neighborhood for social interaction, there exists two Particle Swarm Optimization algorithms, namely global best PSO and local best PSO.

Exploration refers to the ability to explore the solution space in order to find the optimal solution. Conversely, exploitation refers to the ability to concentrate the search in a given promising region in order to improve a candidate solution. An optimization algorithm has to balance between these two terms, since the efficiency and accuracy of the algorithm are determined by the relationship between them [40]. Exploration and exploitation are addressed in the velocity equation. This is done using an additional parameter (i.e. the inertia weight) discussed in Section 3.2.6.

Global Best Particle Swarm Optimization

Global best particle swarm optimization, also referred as gbest PSO, is a variant of Particle Swarm Optimization in which the neighborhood of a particle is the entire swarm. Since all particles communicate together, the topology that gbest PSO uses is a star neighborhood topology. Therefore, the best candidate particle has the largest influence in the entire swarm, and is reflected in the velocity term. Since gbest PSO has larger particle inter-connectivity, it tends to converge fast, but can get trapped in local minima [39].

For any k-dimensional particle i, the j^{th} component of its velocity at time $t+1$, denoted by $v_{ij}(t+1)$, can be obtained by [39]:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[s_{best,j}(t) - x_{ij}(t)] \text{ with, } 1 \leq j \leq k \tag{3.25}$$

where $v_{ij}(t)$ is the j^{th} component of the velocity of particle i at time t, c_1 and c_2 are acceleration constant, $r_{1j}(t)$ and $r_{2j}(t)$ are two random values that are updated with each iteration, $x_{ij}(t)$ is the position of the particle i at time t along dimension j, $p_{best,ij}$ is particle i's best position along dimension j, and $s_{best,j}$ is the swarm's best position along dimension j.

The personal best position for particle i, $p_{best,i}$ refers to the best position that particle i has visited since the first iteration. The personal best position at time t is obtained as [39]:

$$p_{best,i}(t) = \begin{cases} p_{best,i}(t-1) & \text{if } f(x_i(t)) \geq f(p_{best,i}(t-1)) \\ x_i(t) & \text{if } f(x_i(t)) < f(p_{best,i}(t-1)) \end{cases} \tag{3.26}$$

where $f(\cdot)$ is a fitness function.

The global best position, s_{best}, refers to the best position that the swarm has visited since the first iteration. It is defined as [39]:

$$s_{best}(t) \in \{p_{best,0}(t), ..., p_{best,n}(t)\} | f(s_{best}(t)) = min\{f(p_{best,0}(t)), ...f(p_{best,n}(t))\} \tag{3.27}$$

Notice that, the global best position can be calculated as the best personal best position.

Local Best Particle Swarm Optimization

To avoid being stuck in local minima, another variant namely local best PSO. Local Best Particle Swarm Optimization, also referred as lbest PSO, was developed. In lbest PSO each particle is only influenced by the best candidate within the particle's neighborhood. Usually, the implemented network topology for lbest PSO is a ring topology. The velocity term in this case reflects local knowledge within the neighborhood of the particle. Since the particles in lbest PSO exchange information within a neighborhood, PSO tends to converge slow, but is less likely to get trapped in a local minima.

For any particle i, its velocity at time t can be obtained by [39]:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[l_{best,j}(t) - x_{ij}(t)] \tag{3.28}$$

where $l_{best,j}(t)$ is the best position of the neighborhood for particle i along dimension j.

Let N_i be the neighborhood of particle i, the best position of neighborhood N_i, $l_{best,j}(t)$, is defined as [39]:

$$l_{best}(t) \in \{N_i | f(l_{best}(t)) = min\{f(p)\}, \forall p \in N_i\} \tag{3.29}$$

It is important to note that the selection of the particle's neighborhood is based on particles indices. Even though, there exists strategies to create neighborhoods based on spatial similarities, neighborhoods created using the particle's indices are preferred [39]. The two main reasons for this are:

1. It is computationally inexpensive, because it does not require the particles to be ordered based on any specific attribute.

2. It promotes the dissemination of information for all particles, regardless of their current position in the solution space.

Since neighborhoods can overlap, a particle will be a member of multiple neighborhoods. In addition, since neighborhoods are interconnected, they share information and this guarantees that the swarm will converge to a single point, which tends to be the global best particle of the swarm.

Since the only difference between Equation 3.25 and Equation 3.28 is $s_{best}(t)$ and $l_{best}(t)$, respectively, let $B_{best}(t)$ be defined as:

$$B_{best}(t) = \begin{cases} s_{best}(t) & \text{for gbest PSO} \\ l_{best}(t) & \text{for lbest PSO} \end{cases} \tag{3.30}$$

Using $B_{best}(t)$, define a general velocity equation as:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \tag{3.31}$$

Particle Swarm Optimization Parameters

There are several parameters that affect the performance of PSO. The impact of each of these parameters is problem-dependent. In this section, the following parameters are discussed: swarm size, swarm topology, neighborhood size, number of iterations, stopping conditions, velocity components, acceleration coefficients, velocity clamping, inertia weight, and constriction coefficients.

Swarm Size

Swarm size is the number of particles in a swarm. A large swarm size implies a larger initial diversity and allows for larger parts of the solution space to be covered per iteration. While a large number of particles may reduce the number of iterations needed, it increases the computational complexity per iteration which can result in an increase in execution time. The optimal swarm size is problem-dependent, but empirical studies has shown that the swarm size should be between 10 and 60 particles. In the original paper by Eberhart and Kennedy [38], the suggested size for the swarm is 15 to 30 particles, whereas, Bratton and Kennedy [41], suggested a swarm size of 50 particles.

Swarm Topology

Since PSO revolves around social interactions of each particle, the swarm "learns" and adjusts its position towards successful particles in the swarm. The extent of this social interaction is determined by the neighborhood of the particles, and the amount of interaction depends on the size of the neighborhood. For example, smaller neighborhoods have less interaction. As mentioned before, small neighborhoods tend to take longer to converge to a solution, but may return a better solution, while in large neighborhoods the particles converge faster, but may get stuck in local minima. The swarm topology has a strong influence on the performance of PSO. Usually,

neighborhoods are determined by particle indices, but there are other methods to determine the neighborhoods. For example, neighborhoods can be determined based on Euclidean distances between particles [41]. The four most used neighborhood topologies are:

1. Star Topology

   In this topology, each particle can communicate with every other particle in the swarm. This topology tends to lead to faster convergence at the risk of getting trapped in local minima. Each particle tries to imitate the behavior of the best candidate particle present in the swarm. This is the topology implemented in the gbest PSO. For the lbest PSO, if this topology is used, it will have the same local minima problem on the extreme cases, which are when the neighborhood size is either 0 or the swarm size.

2. Ring Topology

   In this topology, each particle can communicate with its n immediate neighbors; for example, for n = 2, any given particle will communicate only with its immediately adjacent neighbors. This topology has a slower convergence but larger portions of the solution space are explored, which leads to better solutions. Each particle tries to imitate the behavior of the best candidate particle within its neighborhood. In this topology, the neighborhoods overlap allowing exchange of information between neighborhoods, which leads the swarm to convergence to a single solution. This is the topology implemented in the lbest PSO case.

3. Wheel Topology

   In this topology, one particle is chosen as the focal particle while all other particles are isolated from one another. The focal particle is connected to every other particle in the neighborhood and all information is communicated through it. The focal particle gathers the performances of all the particles in the neighborhood and adjusts its position towards the best performing particle. Then, the

new position of the focal particle is transmitted to all particles if there is an improvement. This topology tends to slow down the propagation of information in the swarm.

4. Four Clusters Topology

   In this topology, the swarm is divided into four clusters. The clusters are connected by two edges to adjacent neighboring clusters and by one edge to opposite clusters. The particles inside the clusters are usually connected using a star topology or a ring topology.

Figure 3.10 depicts each of the four most used topologies networks.

Neighborhood Size

The neighborhood size impacts the social interaction within the swarm and influences the movement of the particles. For smaller neighborhoods, the algorithm takes longer to converge, but it is more reliable to converge into optimal solutions, since smaller neighborhoods are less vulnerable to local minima. Meanwhile, larger neighborhoods tends to converge faster but at the cost of having some parts of the solution space unexplored. The suggested neighborhood size is about 10-20% of the swarm size [42].

Number of Iterations

The number of iterations needed to obtain good results depends on the problem. A low number of iterations may cause the algorithm to stop prematurely, which would lead to a non-optimal solution. A large number of iterations can add unnecessary computational complexity. Normally, the number of iterations is not the only factor that decides when the PSO algorithm will stop, but it is a criteria that has to be taken into account. Also, if the number of iterations is too large and the stopping criteria

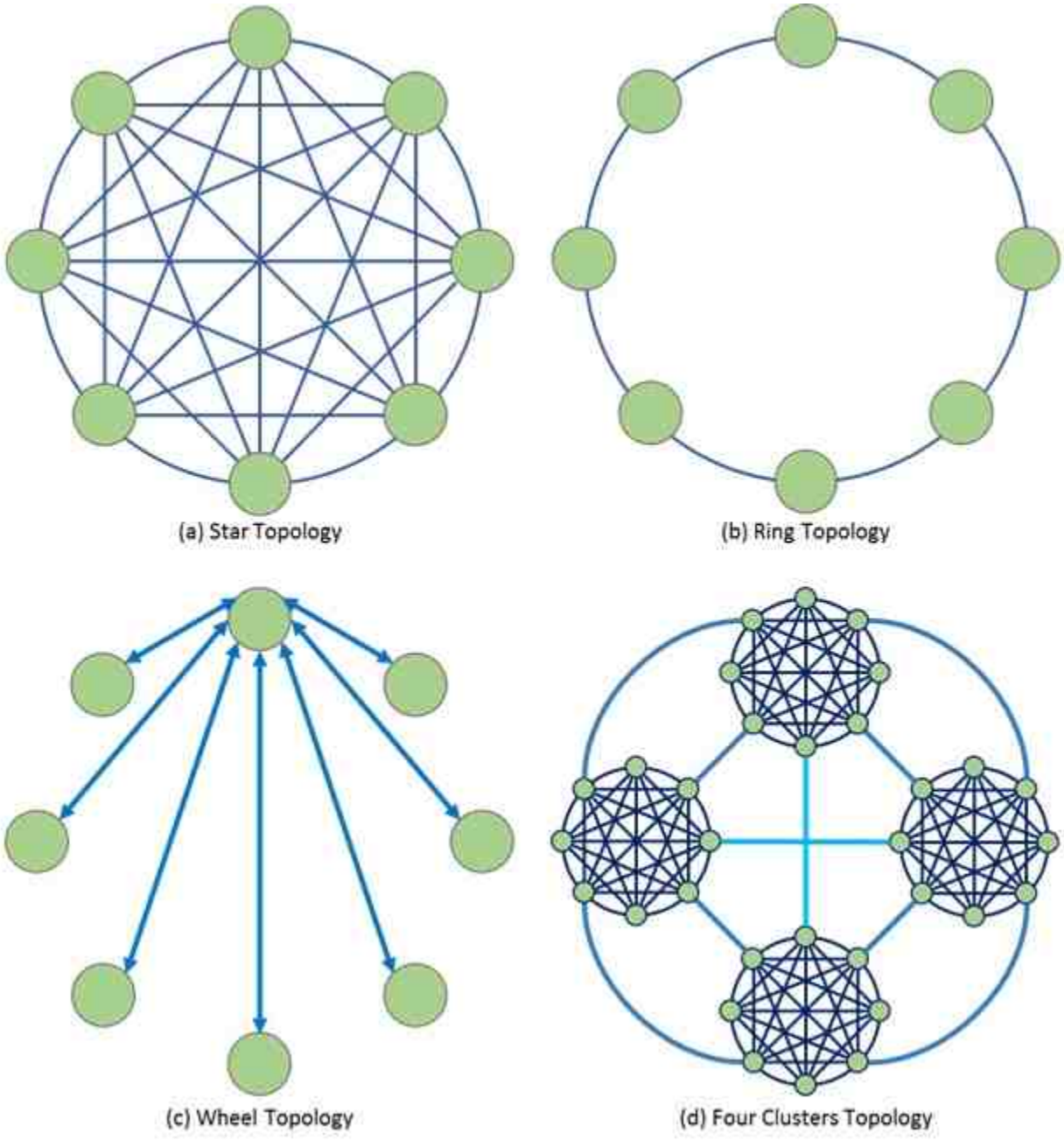


Figure 3.10. Example of neighborhood topologies

is too narrow, the algorithm will start to wander in the solution space. Typically, the number of iterations is set to a high value and the implementation of a stopping condition is used to allow the algorithm to stop sooner.

Stopping Condition

The stopping condition is used to determine if the algorithm has reached an optimal solution to the problem. A good stopping condition does not cause the swarm to converge prematurely. Moreover, it does not create oversampling of the fitness function, since this incurs unnecessary usage of computational resources which in turn can increase execution time. Some stopping conditions proposed in are:

- A certain amount of iterations has been reached.
  The algorithm will stop when a fixed amount of iterations has been executed. It is easy to note that if the specified number of iterations are too small, the algorithm will stop before a reasonable solution has been found. Usually, this condition is used in conjunction with others, as a way to force the algorithm to finish execution in case the swarm fails to converge.

- An acceptable solution has been found.
  The algorithm will stop as soon as a particle finds a solution that is within an acceptable error. The selection of the threshold affects the behavior of the algorithm. If the threshold is too large, the algorithm will stop prematurely, while a too small threshold may cause the algorithm to never stop. To implement this stopping conditions successfully, it is required to have prior knowledge of the optimum solution which normally is not available.

- No improvement is observed over a certain number of iterations.
  The algorithm will stop when there is no significant improvement in the swarm after a certain amount of iterations. The improvement can be judged by monitoring the changes in the particle positions or the particle velocities. If these

changes are below a certain threshold after a specified number of iterations, the algorithm will consider the swarm to have converged.

- The normalized swarm radius is below a threshold.
  The algorithm will stop when the normalized radius of the swarm, R_{norm}, is below a certain threshold. The algorithm will stop when all the particles are centered around the global best position. The selection of the threshold affects the performance of the algorithm. Large thresholds may cause the algorithm to stop prematurely, while small thresholds may cause excessive number of iterations.

Velocity Components

Particle's velocity (Equation 3.31) is composed of three terms: previous velocity, the cognitive term and the social term. The previous velocity term $v_i(t)$, also referred to as the inertia component, is used as the memory of the particle's velocity and direction from the previous iteration. This term can be considered as the particle's momentum and is used to prevent the particle from abruptly changing its direction. Additionally, it creates a bias towards the current direction.

The cognitive term $c_1 r_1(p_{best,i} - x_i)$, also referred as the particle's nostalgia, measures the performance of the particle with its past performances. It represents the particle's individual memory of its best position since the first iteration. The impact of this term on the particles is to draw them back to their own past best positions.

The social term $c_2 r_2(B_{best} - x_i)$ measures the performance of the particle with its neighbors, in the lbest PSO case; or with the swarm, in the gbest PSO case. It represents the group standard that the individuals in the swarm seek to attain. The impact of this term in the particles is to draw them towards the best position found in the particle's neighborhood. The cognitive and social terms are influenced by stochastic values r_1 and r_2, which will be discussed in the following section.

Acceleration Coefficients

The acceleration coefficients are responsible for maintaining the stochastic influence of the cognitive and the social components on the particle's velocity. The acceleration coefficients are composed of four parameters: two constants (c_1 and c_2) and two random values updated with each iteration (r_1 and r_2). The constants c_1 and c_2, known as trust parameters, are non-negative values. The constant c_1 is related to how much confidence a particle in to itself and the constant c_2 is related to how much confidence a particle has in its neighbors. A wrong selection for these values will result in divergent behavior. Based on the different values that c_1 and c_2 can have, there are six different cases:

1. $c_1 = c_2 = 0$

   In this case, the particles will keep flying at their current speed until they reach the solution space's boundary. The particle's velocity is obtained as:

$$\begin{aligned} v_{ij}(t+1) &= v_{ij}(t) + (0)r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + (0)r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \\ &= v_{ij}(t) \end{aligned} \tag{3.32}$$

2. $c_1 > 0$ and $c_2 = 0$

   In this case, the particles are isolated from one other. Since the particles do not trust others particles in the swarm, there is no exchange of information between particles. The particle's velocity is found as:

$$\begin{aligned} v_{ij}(t+1) &= v_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + (0)r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \\ &= v_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] \end{aligned} \tag{3.33}$$

3. $c_1 = 0$ and $c_2 > 0$

   In this case, all particles move towards one single point, which is the best particle in the neighborhood. The particle's velocity is obtained as:

$$\begin{aligned} v_{ij}(t+1) &= v_{ij}(t) + (0)r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \\ &= v_{ij}(t) + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \end{aligned} \tag{3.34}$$

4. $c_1 = c_2 \neq 0$

   In this case, all particles move towards the average of the personal best position and the neighborhood's best position.

5. $c_1 >> c_2$

   In this case, a particle is more influenced by its own good performances, which may cause excessive wandering.

6. $c_1 << c_2$

   In this case, the particles are more influenced by the best neighbor particle, which can lead the particles to move towards the local optima prematurely.

Also, small values for c_1 and c_2 result in smooth particle movement. This allows the particles to roam the region in which they are before being attracted towards the "good" regions. Meanwhile, large values cause abrupt movements towards these "good" regions, limiting the exploration of the solution space. The values for c_1 and c_2 are problem-dependent, and normally, they are static values. For example, Kennedy and Eberhart [38] suggest that the values for the two acceleration constants should be $c_1 = c_2 = \sqrt{2}$, and Khan and Engelbrecht [43] suggest that the values should be $c_1 = c_2 = 1.49$.

Velocity Clamping

In a particle swarm optimization algorithm, the velocity value tends to grow quickly. This is more noticeable when the particle's current position is far from its personal best position and the neighborhood best position. Therefore, the particles will have large position changes which can cause the particles to leave the solution space. The aim of velocity clamping is to maintain the particles within the boundaries of the solution space and to guarantee that the size of the steps are reasonable to comb the solution space. To implement velocity clamping, if the velocity of a particle exceeds a specified maximum value, then the particle's velocity is set to this maximum value. The particle velocity is adjusted before updating the position of the particle. (See Equation 3.35) [39].

$$v_{ij}(t) = \begin{cases} v'_{ij}(t) & \text{if } v'_{ij}(t) < V_{max,j} \\ V_{max,j} & \text{if } v'_{ij}(t) \geq V_{max,j} \end{cases} \tag{3.35}$$

where $v'_{ij}(t)$ is found using Equation 3.31.

Velocity clamping depends on the value of $V_{max,j}$. A poorly chosen value leads to poor performance since large values of $V_{max,j}$ favor global exploration while small values of $V_{max,j}$ favor local exploitation. This implies that, if the value of $V_{max,j}$ is too small, the swarm will not sufficiently explore regions beyond the promising regions, and in the worst case, the swarm may get trapped in a local optimum. On the contrary, large values of $V_{max,j}$ have the risk of missing promising regions since the particles are moving erratically and faster.

Even though, the advantage of using velocity clamping is that the changes in velocity are controlled, one disadvantage of velocity clamping is that it affects the direction in which the particles move in addition to the step size. Moreover, if all velocities are equal to $V_{max,j}$, the particles will search only on the boundaries of a hyper-cube defined by $[x_i(t) - V_{max}, x_i(t) + V_{max}]$. This issue can be solved by implementing a method to reduce the value of $V_{max,j}$ over time.

Inertia Weight

The goal of the inertia weight is to control the exploration and exploitation of the swarm. Even though, it was created to overcome some of the disadvantages of velocity clamping, it does not eliminate the need for it. The inertia weight works by weighting the influence of the previous velocity on the current velocity. Using the inertia parameter, the velocity Equation 3.31 changes to [39]:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \quad (3.36)$$

where w is the inertia weight.

Based on the different values that the inertia weight can have, there are three possible different cases:

1. $w = 0$

   For this case, the particles will move without taking into account the previous velocity. Replacing $w = 0$ on Equation 3.36, leads to:

$$\begin{aligned} v_{ij}(t+1) &= (0)(v_{ij}(t)) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \\ &= c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)] \end{aligned} \quad (3.37)$$

2. $w < 1$

   For this case, the particles velocities tend to decrease. Also, this allows the particles to accept quick changes in directions.

3. $w \geq 1$

   For this case, the particle velocities tends to increase over time towards the maximum velocity. This may cause the swarm to diverge. Also, it is more difficult for the particles to change their direction and move back into promising regions.

Based on the behavior of the particles depending on the value of w, it is to be noted that large values of w favor global exploration, while small values of w favor local exploitation. However, for excessively large values the swarm may diverge and for excessively small values, the swarm will have a limited exploration ability. Also, for small values of w, the cognitive and social components have a greater influence in a particle's position.

The inertia weight can be implemented using a fixed or dynamic value. Several researchers favor dynamically changing values because this allows the particles to favor exploration at the early stages while favoring exploitation in later iterations. The basic idea behind these approaches is to start with large inertia values, which will allow the particles to explore the solution space freely, and then decrease them over time, as this favors local exploitation. Van den Bergh and Engelbrecht showed that in order to avoid cyclic or divergent behaviors, and to guarantee convergence in the particles trajectories, the inertia weight has to fulfill the following condition [39]:

$$w > \frac{1}{2}(c_1 + c_2) - 1 \tag{3.38}$$

There are several approaches to implement the concept of dynamic inertia weight. One simple approach is to randomly select a new inertia weight value at each iteration. For example, let the inertia weight be, at each iteration, the sum of the acceleration coefficients [44]. Another implementation is to linearly decrease an initially large inertia weight. For example, let the inertia weight start at the value of 0.9 and decrease until it reaches the value of 0.4 [45]. Also, the inertia weight can be decreased non-linearly over time [39]. Normally, these methods make the particles spend less time in the exploration mode while allowing them to spend more time in exploitation mode.

There are three nonlinear methods for changing the inertia weight:

- Peram, Veeramachaneni and Mohan [45] proposed:

$$w(t+1) = \frac{(w(t) - 0.4)(n_t - t)}{n_t + 0.4} \tag{3.39}$$

  with $w(0) = 0.9$.

- Venter and Sobieszczanski-Sobieski [46] suggested:

$$w(t+1) = \alpha w(t') \tag{3.40}$$

where $\alpha = 0.975$ and t' is the last iteration where the inertia last changed. The inertia weight only changes if the variation of the fitness value is small. The initial value of the inertia weight is $w(0) = 1.4$ and the lower bound is $w(n_t) = 0.35$.

- Clerc [39] proposed:

The inertia weight value should be proportional to the improvement of the swarm, as is obtained by:

$$w_i(t+1) = w(0) + (w(n_t) - w(0))\frac{e^{m_i(t)} - 1}{e^{m_i(t)} + 1} \tag{3.41}$$

where: $w(0) < 1$, $w(n_t) \approx 0.5$, and m_i is the relative improvement that is estimated by:

$$m_i(t) = \frac{f(l_{best,i}(t)) - f(x_i(t))}{f(l_{best,i}(t)) + f(x_i(t))} \tag{3.42}$$

Clerc's notion is that as an individual improves more over its neighbors, it should be able to follow its own path.

Constriction Coefficient

A constriction coefficient, normally represented as χ, was introduced by Maurcie Clerc in [47]. It was developed to eliminate the need for velocity clamping while guaranteeing convergence to a stable point. It works by reducing the velocity at every time step. The velocity equation with the constriction coefficient is given by [39]:

$$v_{ij}(t+1) = \chi[v_{ij}(t) + c_1 r_{1j}(t)[p_{best,ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[B_{best}(t) - x_{ij}(t)]] \tag{3.43}$$

where χ is defined as:

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi(\phi - 4)}|}, \text{ where, } \phi = \phi_1 + \phi_2,\ \kappa \in [0, 1] \text{ and } \phi_i = c_i r_i \tag{3.44}$$

The parameter κ is used to control the exploration and exploitation of the swarm. The conditions for κ and ϕ to guarantee convergence are [39]: $\phi \geq 4$ and $\kappa \in [0, 1]$. Many researchers consider the constriction coefficient approach to be equivalent to the inertia weight approach to the extent that it is possible to obtain an inertia weight model given χ [39], [48]. The equivalent inertia model for a given χ can be obtained using Equations 3.45 and 3.46.

$$w = \chi \tag{3.45}$$

$$\phi_i = \chi c_i r_i, \text{ where, } i = 1, 2 \tag{3.46}$$

General PSO

A particle swarm optimization method can be divided into three important steps: initialization, function evaluation and stopping condition. Particle swarm optimization works by evaluating a function iteratively until a stopping condition is met. The evaluated function is usually called fitness function. A general PSO algorithm is presented below:

PSO Algorithm

Initialize the n particles

Repeat

For each particle

Update the particle's velocities and the particle's position

Evaluate the particle's fitness formula

Check if the new particle position has a better result than lbest

End the for loop

Check if any of the new particles positions has a better result than the gbest.

Until the stopping condition is met.

Return the global best position as the solution.

End

Initialization Step

The first step of the particle swarm optimization algorithm is the initialization step. In this step, the swarm and control parameters are initialized. The constants, such as the acceleration coefficients c_1 and c_2, the swarm size and the neighborhood size, are specified. If a fixed inertia weight and fixed maximum velocity are used, they need to be specified in this step.

The initial swarm has an impact on performance since a bad initial swarm can not cover the entire solution space and it will be harder to find a solution that lies outside of the initial covered area. Usually, the particles are initially uniformly distributed over the solution space. Some researchers favor the use of a completely random distribution, but the issue emerges by doing this is that the swarm can have some bias towards certain regions due to an increased number of particles present over those regions. On the other hand, uniformly distributed particles do not guarantee good performance.

Usually, the initial velocities of the particles are set to zero, but, depending on the task in hand, these velocities may be set to non-zero values, while taking into consideration that large velocities will generate large position updates, and these large updates may cause the particles to move outside of the solution space boundaries.

The particle's personal best position is initialized to its current value. This means, the particle's personal best position at time $t = 0$ is equal to the particle's initial value. The best global particle position is initialized to the initial particle position value which has the lowest fitness function value at time $t = 0$.

Function Evaluation

The function evaluation step consists of the actions that the algorithm must perform during each iteration until one or more of the stopping condition are satisfied. Usually, this step is the one that consumes more time and resources during the execution of a PSO algorithm. The runtime of the function evaluation step depends of the number of particles in the swarm. In one iteration, the algorithm must perform the following actions for each particle in the swarm:

1. Find the velocity along each dimension.
2. Update the particle's position.
3. Evaluate the fitness function.
4. Determine the particle's personal best position.
5. Determine the local best position, in lbest PSO, and the global best position, in gbest PSO.

3.2.7 Microsoft Kinect

Microsoft Kinect is an input device used for motion sensing that enables the user to interact with a system using gestures and spoken commands. Its software technology has been developed by Rare, a subsidiary of Microsoft since 2002, while the camera technology has been developed by PrimeSense, a subsidiary of Apple since 2013. The first version of the Microsoft Kinect was introduced in November 2010 and its upgraded version was introduced in November 2013. The Microsoft Kinect was originally introduced as an accessory for the Xbox 360, but now, because of the information that the Microsoft Kinect is able to acquire, it is not limited to gaming and is a tool used in different fields in industry and research, such as health-care, image processing, computer vision, robotics, virtual reality, and security system.

Figure 3.11. Microsoft Kinect Sensor.

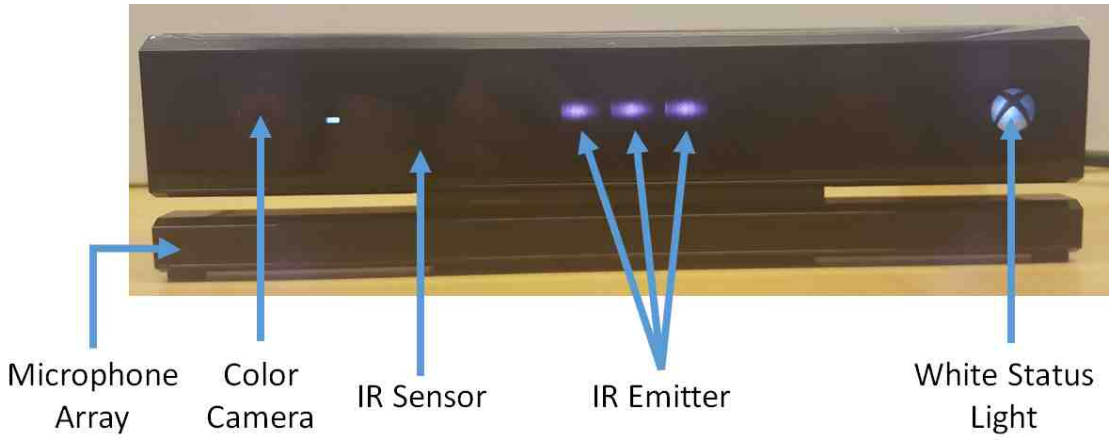


Figure 3.12. Components of the Microsoft Kinect Sensor.

Hardware Components

A Microsoft Kinect sensor is composed of a four-microphones array, a color camera, an IR emitter, and an IR sensor.

Color Camera

The Kinect Sensor is equipped with a color camera that is able to capture video that has a resolution of 1920x1080 pixels at a frame rate of 30 fps. Also, it works in three different color formats: RGBA, GBRA and YUV2. Some features that the camera possess are: white balancing, black reference, flicker avoidance, and color saturation. The field of view for the color camera is 85° horizontal and 54° vertical, as shown in Figure 3.13.

IR Emitter and IR depth Sensor

The IR emitter and the IR depth sensor are used to generate the infrared and depth images. The IR emitter is an IR projector that emits infrared light in a "pseudo-random" speckle pattern onto the scene. The light reflected from the speckles is captured by the IR sensor which converts it into depth and infrared information. The

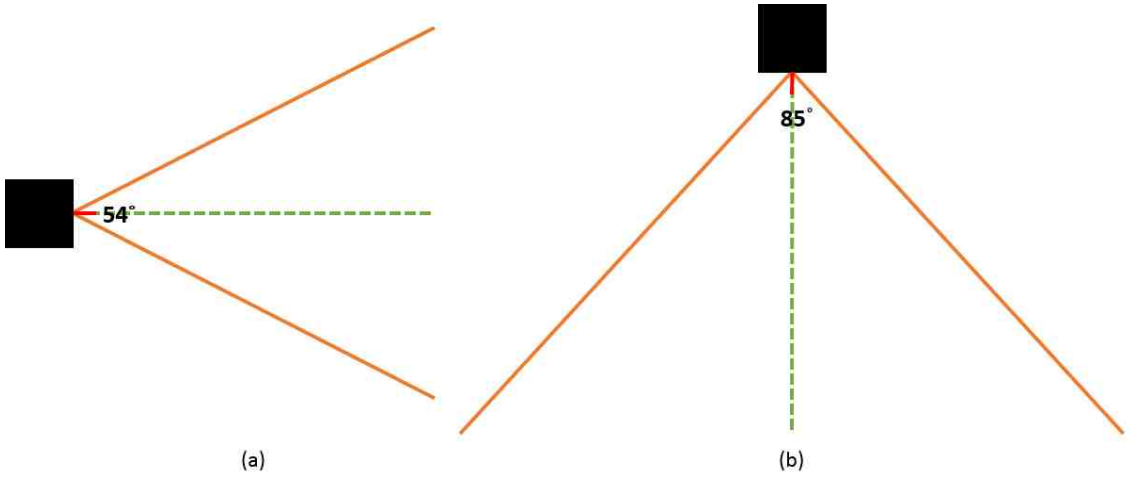


Figure 3.13. Field of view of the color camera.

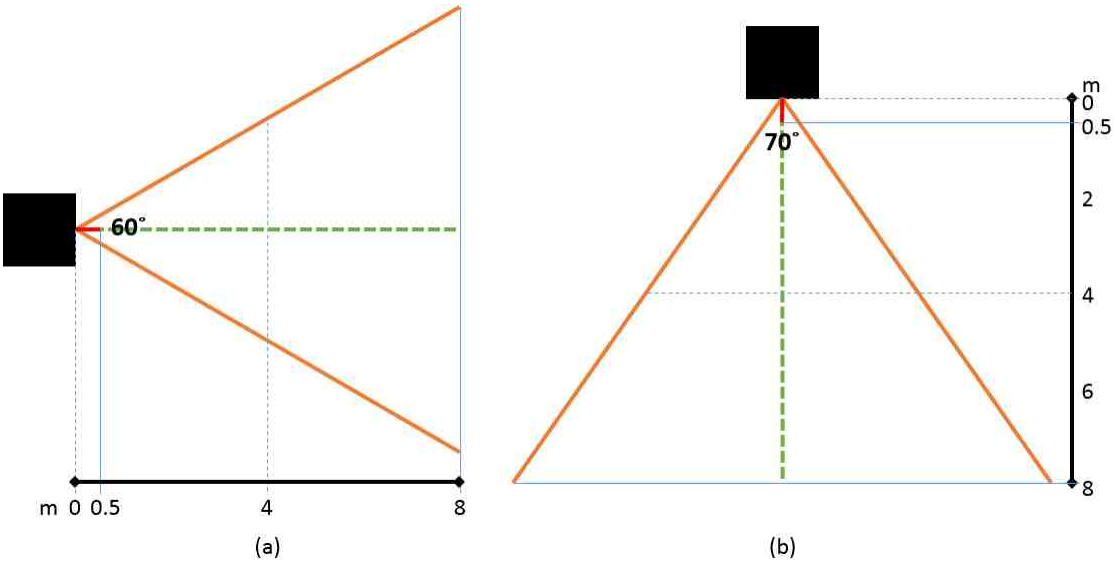


Figure 3.14. Field of view of the IR Emitter/IR Sensor.

IR sensor can work regardless of the lighting conditions of the scene. The resolution of both images is 512x424 at a frame rate of 30 fps. Each pixel in an infrared frame has a 16-bit value to represent the IR intensity. On the other hand, each pixel value in the depth map represents a distance in millimeters that ranges from 500 mm to 8000 mm, which is the working range of the IR sensor. The field of view for the IR sensor emitter is 70° horizontal and 60° vertical, as shown in Figure 3.14.

Infrared and Depth Images

Microsoft Kinect uses its IR emitter and IR sensor to generate real-time depth maps and infrared frames. The resolution of the depth maps and the infrared frames is 512x424 at a frame rate of 30 fps. To generate these frames, the Kinect sensor employs two techniques: Structure light and Time of Flight (TOF). Structure light is a method of sending a known light pattern, usually a grid or horizontal bars onto a scene. Using the way the pattern deforms when hitting the surface of objects in the scene, allows the on board processor to calculate the depth and surface measurements of the object. Similarly, TOF is the process of measuring the time it takes for an

Figure 3.15. Example of an infrared image.

object to travel a distance through a medium. The pattern used by the Kinect sensor for structure light is a speckle pattern, and the infrared frames are generated by capturing the intensity of infrared light that is reflected.

To generate depth maps, the IR sensor measures the time used by infrared light to leave the sensor and return to it. Using this time, the Kinect calculates the distance between the sensor and the corresponding object which the infrared light reflected off. Depth maps can be used to generate a set of discrete 3D data points, called a point cloud. It is necessary to emphasize that the Kinect sensor is innately noisy, which causes fluctuations in the depth measurements and as a result may create "hot" or "cold" spots in the depth maps where no measurements can be obtained.

Figure 3.16. Example of a depth map.

Figure 3.17. Example of a point cloud.

Kinect Software Development Kit

The Microsoft Kinect has a software development kit (SDK) which grants users the ability to develop their own applications using the Microsoft Kinect sensor as a sensing device. This toolkit is comprised of a set of libraries that provide an interface for communication with the Microsoft Kinect sensor. It also includes the application program interface (API) and the system drivers for the sensor. Applications can be built using C++, C# and Visual Basic. Some of the functions included in the Kinect SDK are: skeleton recognition, skeleton tracking, facial tracking and speech recognition. In addition to these basic functions, the Kinect SDK contains: a face API, which facilitates the creation of applications that require motion tracking and detection of human faces, Kinect Studio, which allows recording and playback of the acquired depth and color data, and documentation. The minimum system requirements that must be meet to be able to use the Microsoft Kinect and the Kinect SDK are:

- The operating system has to be: Windows 8, Windows 8.1, or Windows 10.
- A 64-bit physical dual-core 3.1 GHz or equivalent processor.
- 4 GB of RAM.
- A graphics card that supports DirectX 11,
- A USB 3.0 host controller.

3.3 System Workflow

The proposed motion correction system consist of four sequential stages:

- The first stage is responsible for capturing and storing the infrared and depth images from the Microsoft Kinect. This stage works in parallel while performing PET/CT scanning. Ideally, the Kinect sensor will start capturing images at the same time the scanner is started, and will stop when the scanning is completed.

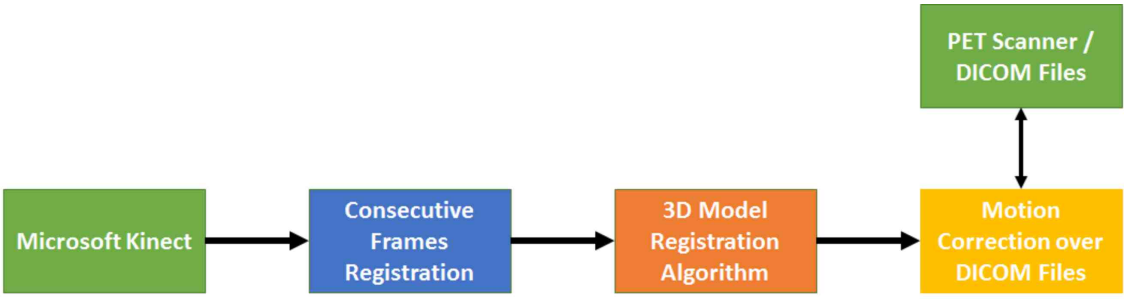


Figure 3.18. Block Diagram of the proposed system.

- The second stage is in charge of performing the registration between consecutive frames. It is composed of three steps: a Kinect file reader step, a features extraction and matching step, and a particle swarm optimization step.

- The third stage consists of the registration of all frames with respect to a reference frame. This stage returns the affine transformation matrices with their corresponding time stamps.

- The purpose of the fourth stage is to perform motion correction on the DICOM files from the PET/CT scanner using the affine transformation matrices obtained from the previous stage.

The system has the following inputs: the infrared and depth images from the Microsoft Kinect, as well as the DICOM files from the PET/CT Scanner. The final output of the system will be motion corrected images contained in corrected DICOM files. The system does not require markers for registration and tracking of the patient movements. Figure 3.18 shows a block diagram that illustrates the proposed motion correction system. The following sections of this chapter explain in detail each element of the block diagram presented in Figure 3.18.

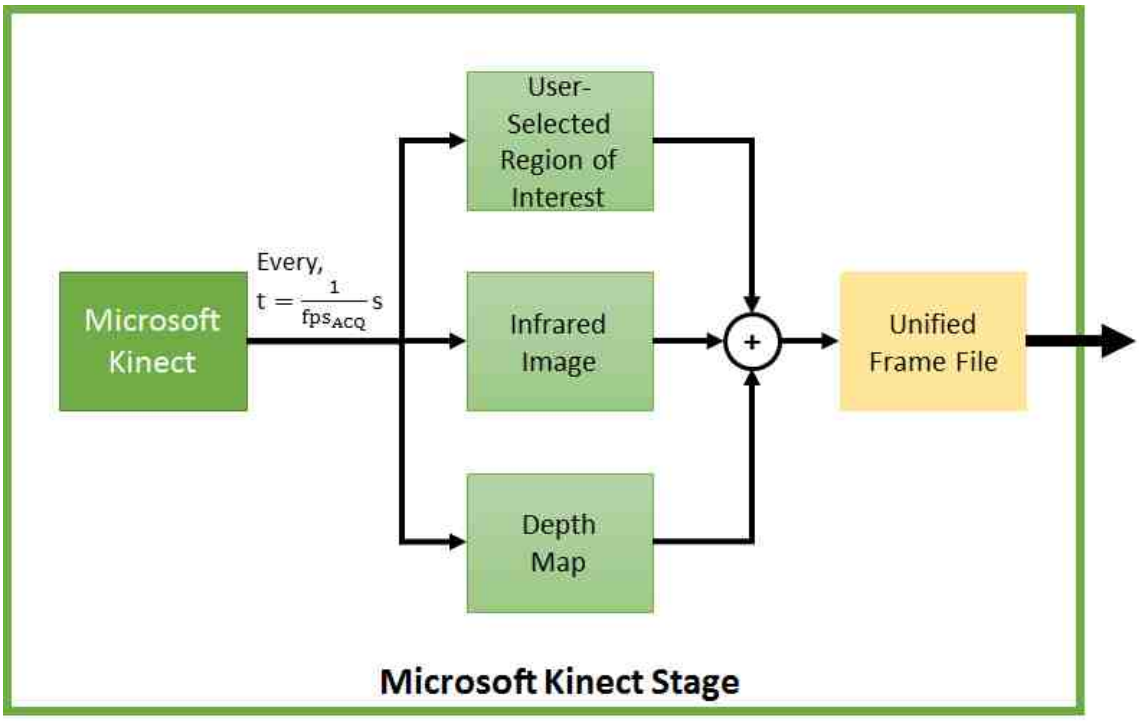


Figure 3.19. Microsoft Kinect Stage.

3.3.1 Microsoft Kinect Stage

This stage consists of algorithms necessary for the Microsoft Kinect to capture and store the infrared and depth images of the patient inside a PET/CT scanner during the examination. The Microsoft Kinect sensor and a user-defined region of interest (ROI; used in the next stages) are the two inputs for this stage. The output is a file which contains the infrared and depth information as well as the information of the ROI defined by the user. These files are saved locally for the posterior motion correction stages. Figure 3.19 shows a block diagram that describes this stage.

The initialization and termination processes of the Kinect sensor are handled using functions found in the Kinect SDK. In addition, SDK functions are used to extract the infrared and depth maps arrays from the sensor. Even though the Kinect is able to work at a maximum frame rate of 30 fps, the acquisition frame rate used in this

project is lower than this, as explained below. Additionally, instead of creating two files at every acquisition time step (one for the infrared frame and one for the depth map) these two files are combined into one single file which is the input file used in the following stages.

In order to choose the appropriate frame rate, some limitations must be considered:

1. A PET scan can take between 15 to 60 minutes (or 900 to 3600 seconds). If the acquisition frame rate is 30 fps and assuming that each frame has a size of 1 MB, the total amount of saved information can be computed by:

$$\text{dataSize} = t_{scan} * \frac{30 \text{ frames}}{\text{s}} * \frac{1 \text{ MB}}{\text{frame}} \tag{3.47}$$

   where t_{scan} is the duration of the PET scan in seconds. Using Equation 3.47, the amount of acquired data for a 60 minutes scan will be 105.47 GB.

2. In practice, the size of each of frame file is 4.27 MB. Replacing the real file size in Equation 3.47 gives a amount of data equal to 450.35 GB for a 60 minutes PET/CT scan.

3. The hard disk drive write speed, and the Kinect algorithm running time create a bottleneck for the system. One approach that helps overcome this issue is to limit the amount of information that will be stored.

4. The amount of data generated during this stage determines the duration of the subsequents stages.

Hence, there must be a balance between the acquisition frame rate, the execution time of the algorithm, the size of the acquired data, and the ability to detect the patient's movements. The machine used for the development of this project allows to store up to 2 frames every second without compromising the speed of the system.

The infrared images and the depth maps are acquired from the Kinect as two arrays, where each entry in the array is pixel value in the corresponding image. The Kinect creates both images simultaneously and with the same resolution. The images

are created using the same mechanisms and sensors described in Section 2.4. The unified frame file is then created by simultaneously reading the same position in each array, and the values written in the corresponding entry of the output file. The creation of a single combined file facilitates the handling of the information in the subsequent stages. This file consists of: one entry for the region of interest and 217088 entries for the infrared and depth data (this value is based on a frame resolution of 512x424). The region of interest entry consists of the x and y coordinates of the upper left corner of the region of interest, as well as the width and the height of the region of interest. Each infrared/depth entry has four parameters:

1. The location along the x-axis, which varies from 0 to 511.
2. The location along the y-axis, which varies from 0 to 423.
3. The infrared pixel value, which varies from 0 to 65535.
4. The depth value, which varies from 0 to 8000.

3.3.2 Consecutive Frames Registration Stage

The purpose of this work is to correct artifacts arising from patient motion. An issue that arises is that in order to perform the registration between frame i and the initial frame, it is necessary that the previous frames be already registered against the initial frame. Therefore, this forces the system to register each frame sequentially which can make the execution time of the algorithm impractical. In addition, the number of files generated by the Microsoft Kinect stage must be considered, because the number of files have a large impact on the execution time of the registration stage. If the number of files generated at that stage is excessively large, the execution time of the registration stage will be too long which could make the use of the proposed system unfeasible.

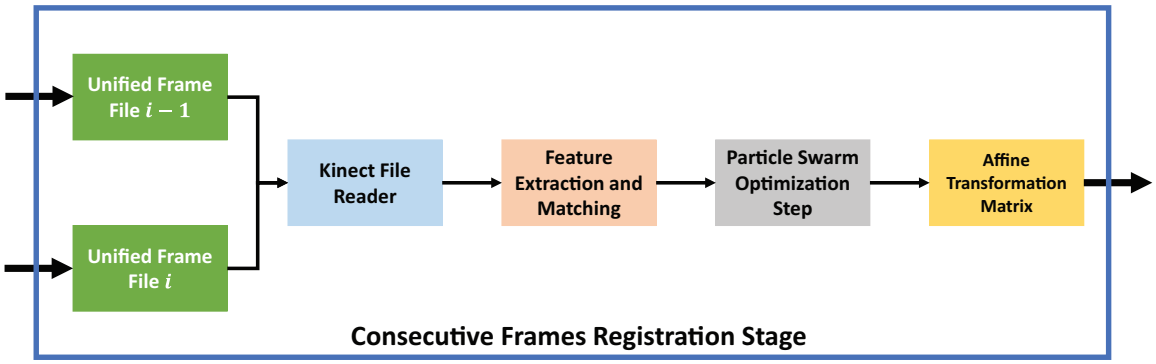


Figure 3.20. Consecutive Frames Registration Stage.

One way to reduce the runtime of the system is to parallelize the frame registration process. Registering two consecutive frames does not require knowledge of previous results, hence, the proposed system divides the registration process into two stages: the process of registration between consecutive frames and the registration of all the frames to an initial reference frame.

It should be noted that there is no prior information linking two consecutive frame files. In addition, the only relationship between the infrared images and the depth maps is that the objects present in the scene occupy the same position in both images. To perform registration in 3D, the proposed system requires correspondences between the entries of the point clouds. Feature extraction and matching is used to create these correspondences. Moreover, these techniques are applied to the infrared images since they have better contrast and more features than the depth maps.

This stage takes as input the n frame files generated in the Kinect Stage. Its output consists of $n-1$ affine transformation matrix, which correspond to the maximum number of possible combinations of consecutive files. This is accomplished in three steps. The first step takes the Kinect frame files and generates the infrared images and the point clouds of the region of interest. The second step uses the infrared images and the point clouds to generate a pair of matched arrays which will be used to perform the registration. The third step performs the affine registration using

PSO over the matched arrays. Since the algorithm only requires the two consecutive frame files to obtain the affine transformation matrix that register both files, multiple registration processes can be run in parallel, leading to an improvement in the run time of the system. Figure 3.20 illustrates the process performed in this stage.

Kinect File Reader

The purpose of this step is to read the unified frame file created in the Microsoft Kinect stage. From this file, the infrared image and the point cloud of the region of interest that was defined by a user during PET examination are created. Figure 3.21 shows the implementation of this step.

The region of interest data is used to create a mask. This mask is a 512x424 image that consist of 0's and 1's, where each pixel with a value of 1 implies that it belongs to the region of interest, whereas if it is 0 it does not belong to the region of interest. Since the region of interest has a rectangular shape, the only information needed to create it is: the x and y coordinates of one of its vertices (x_{ROI} and y_{ROI}), the width (w_{ROI}), and the height (h_{ROI}). As described Section 3.2.1, the frame file contains this information. To create the mask, the algorithm iterates through each pixel of the mask. If the pixel coordinates in x is between x_{ROI} and $x_{ROI} + w_{ROI}$ and its coordinate in y is between y_{ROI} and $y_{ROI} + h_{ROI}$ then, the value is set to 1, otherwise, it is set to 0. This process is performed only if the ROI information changes.

To generate the infrared images, for each entry of the frame file, the read infrared value is placed in its corresponding pixel position, then, the infrared image is multiplied by the mask. This process will extract the region of interest in the infrared image. The point clouds are generated using the depth maps, where depth values vary from 0 to 8000, representing the distance, in centimeters, of the Kinect sensor to the objects in the scene. The depth map is multiplied with the mask to extract the region of interest. Each pixel in the region of interest in the depth map generates

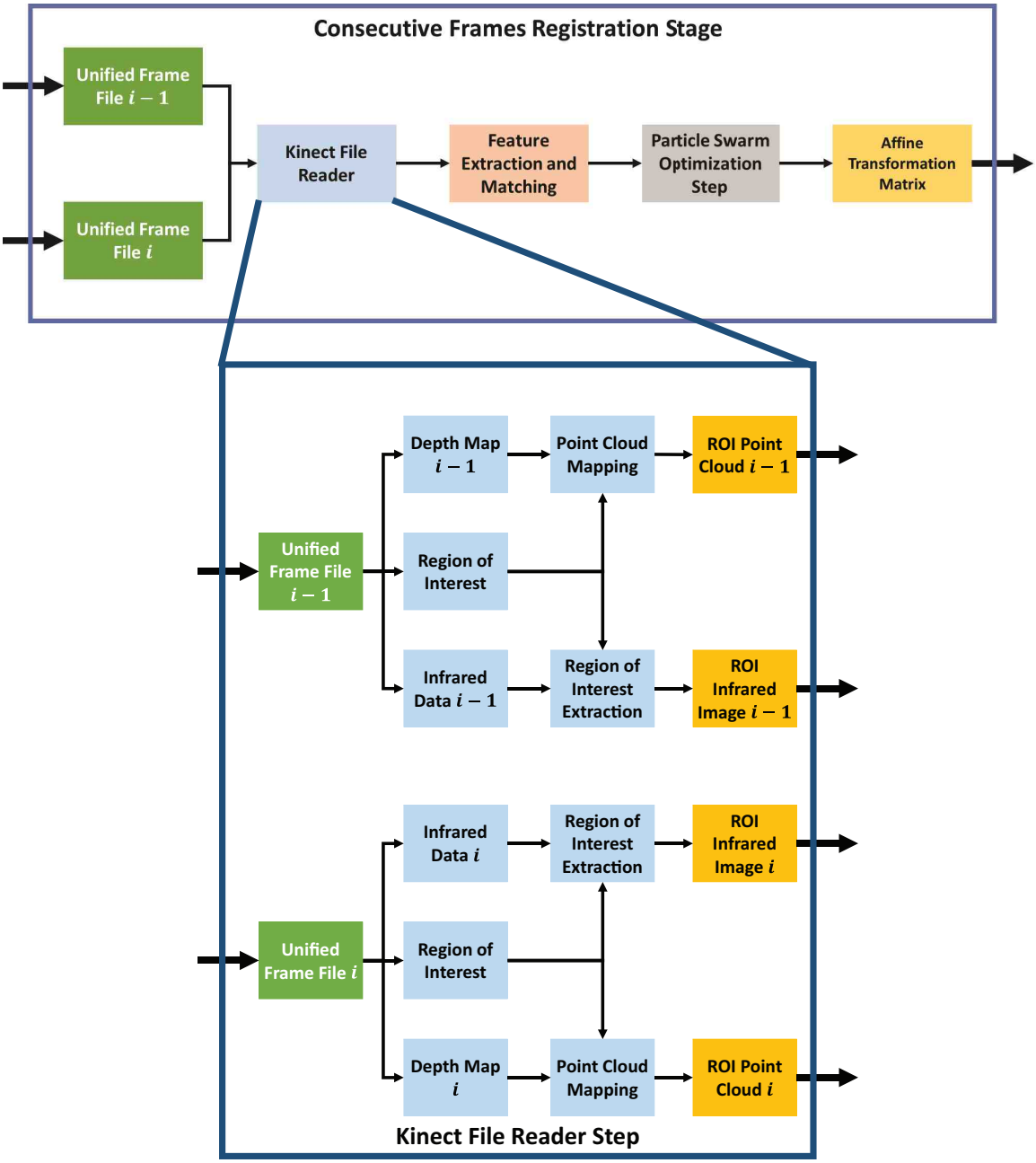


Figure 3.21. Kinect File Reader Step.

a point in the point cloud. Using equations 3.48, 3.49 and 3.50, the 3D coordinates of these point are obtained. Each entry of the point cloud is linked to its corresponding pixel in the infrared image.

$$x_i = \frac{u_i - c_x}{f_x} * pixel_{u,v} \tag{3.48}$$

$$y_i = \frac{v_i - c_y}{f_y} * pixel_{u,v} \tag{3.49}$$

$$z_i = pixel_{u,v} \tag{3.50}$$

where: u_i and v_i are the x and y coordinates of the i-th pixel in the depth map, $pixel_{u,v}$ is the value of the i-th pixel in the depth map, f_x and f_y are the horizontal and vertical focal length of the Kinect sensor, c_x and c_y are the location of the center point of the Kinect, and x_i, y_i and z_i are the 3D coordinates of the i-th entry of the point cloud. The Kinect SDK allows the user to obtain the values of f_x, f_y, c_x and c_y.

Feature Extraction and Matching Algorithm

The frame files generated by the Kinect do not have any relationship between them. It is necessary to create a relationship by taking the infrared images and the point clouds to generate a pair of matched arrays. These arrays are the input information used by the PSO step to obtain the affine transformation matrix that aligns the point clouds. The most important consideration in the development of this project is that the lightning conditions in the room may vary. For these reason, it was decided to use the infrared camera instead of the RGB camera.

In this step SURF is used to detect and extract the features in the infrared images. Once the features are extracted, the system proceeds to obtain the matches between both images. The output is the pair of matched point cloud arrays as shown in Figure 3.22.

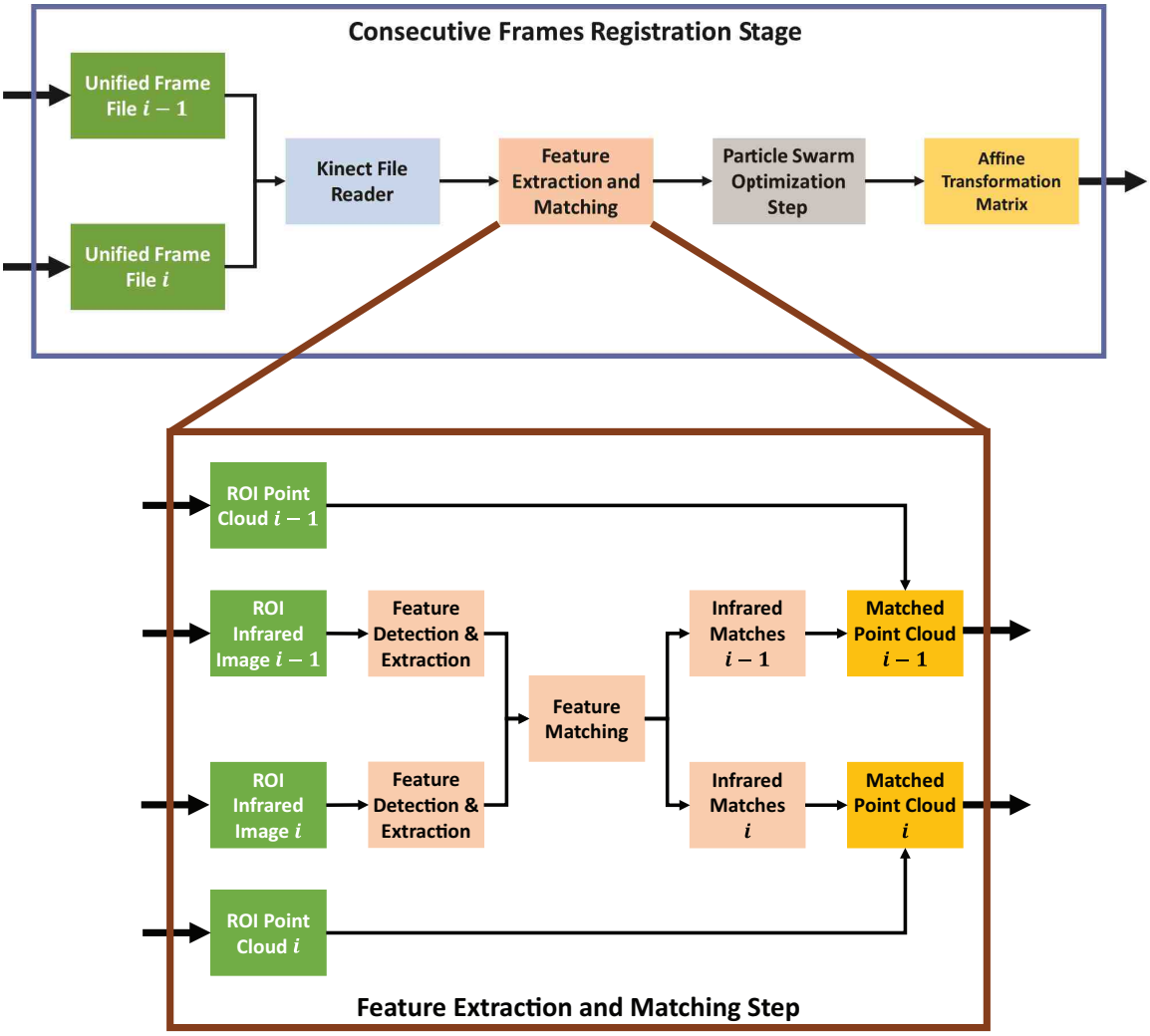


Figure 3.22. Features Extraction and Features Matching Step.

Feature matching is done using a nearest neighbor search library called Fast Approximate Nearest Neighbor Search (FLANN), which is implemented in OpenCV. FLANN takes two features arrays (source and reference) and creates a k-d tree for each array. A k-d tree is a data structure used to arrange the points in a k-dimensional space. These structures are useful for applications that involves a multidimensional search key, such as in nearest neighbor searches. To perform the feature matching process, FLANN takes a feature of the source array and it finds its nearest neighbor in the reference array by performing a query in the k-d tree. This process is repeated for each feature in the source array. The matches returned by FLANN are passed through an outliers removal step. Since it is assumed that between consecutive frames there is little movement, if the distance between two matched features is considerably large, the match is considered an outlier. To decide which matches will be removed, the mean and the standard deviation of the distances of all the matches are obtained. All matches whose distance is larger than the mean plus the standard deviation are removed. The remaining matches are used in the next step. This is described in Equation 3.51.

$$O(M_i) = \begin{cases} true & \text{If } d_i \geq \overline{d} + \sigma_d \\ false & \text{Otherwise} \end{cases} \tag{3.51}$$

where $O(M_i)$ is the function that determines if the matched features i is an outlier, F_i is the i^{th} matched feature, d_i is the distance between features, $\overline{d}$ is the mean of the distances of all the matches, and σ_d is the standard deviation of the distances of all the matches.

The last step of this stage is dedicated to the creation of arrays used by the PSO algorithm. Each element of the matched features array, created in the previous step, represents a match and consists of two entries. The first entry is the location of a feature in the source infrared image, while the second entry is the matched feature in the reference image. To generate the matched point clouds, the feature entries in the matched features array are located in their respective depth maps, and nxn square

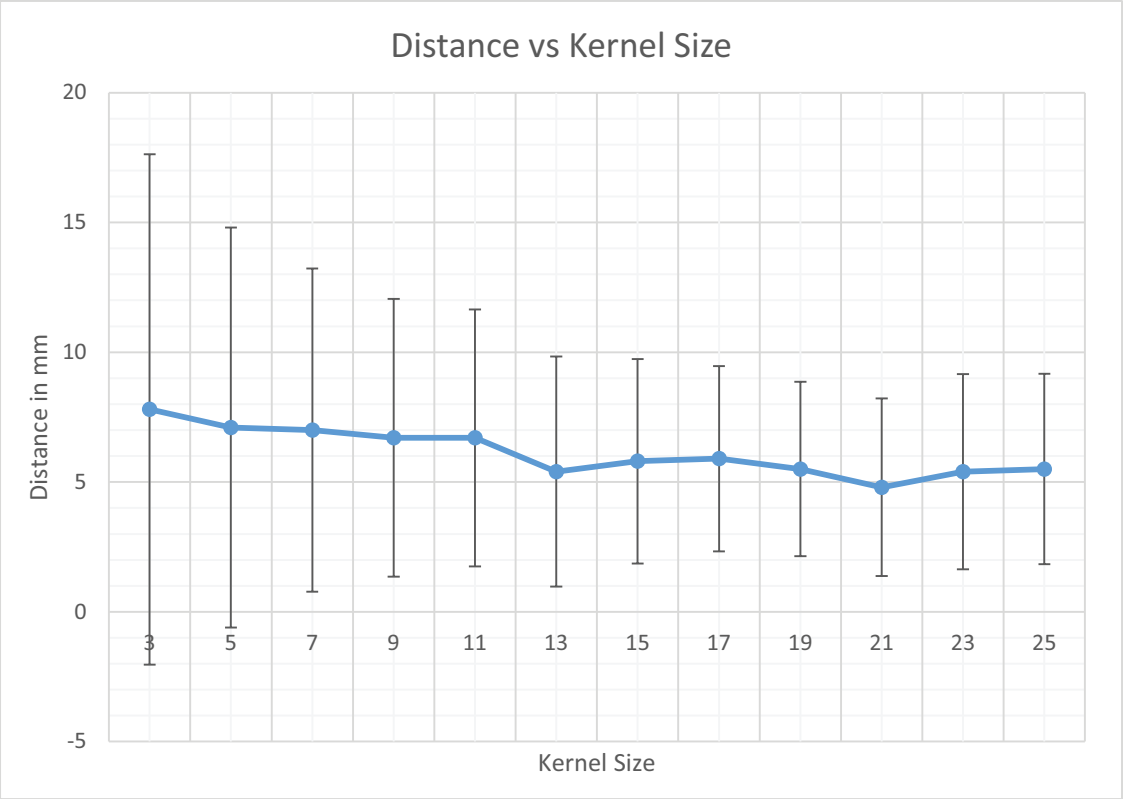


Figure 3.23. Obtained average distance and standard deviation of the matches at the output of the system while varying the kernel size.

kernel placed at each coordinate. All distance values within the kernel are averaged. Using the mean distance and the 2D coordinates of the feature, the 3D coordinates can then be obtained using Equations 3.48, 3.49 and 3.50. These coordinated values represent the corresponding matched feature in the point cloud, are subsequently saved in the same location on the output point clouds arrays. The process is repeated with each element of the matched features array and the output generated consists of two point cloud arrays whose entries represent the matched features of the infrared images in 3D. The kernel size was chosen based on tests were performed using the complete system while the kernel size value was varied. Figure 3.23 shows the results obtained from these tests. In this case the chosen value for the kernel size is 21.

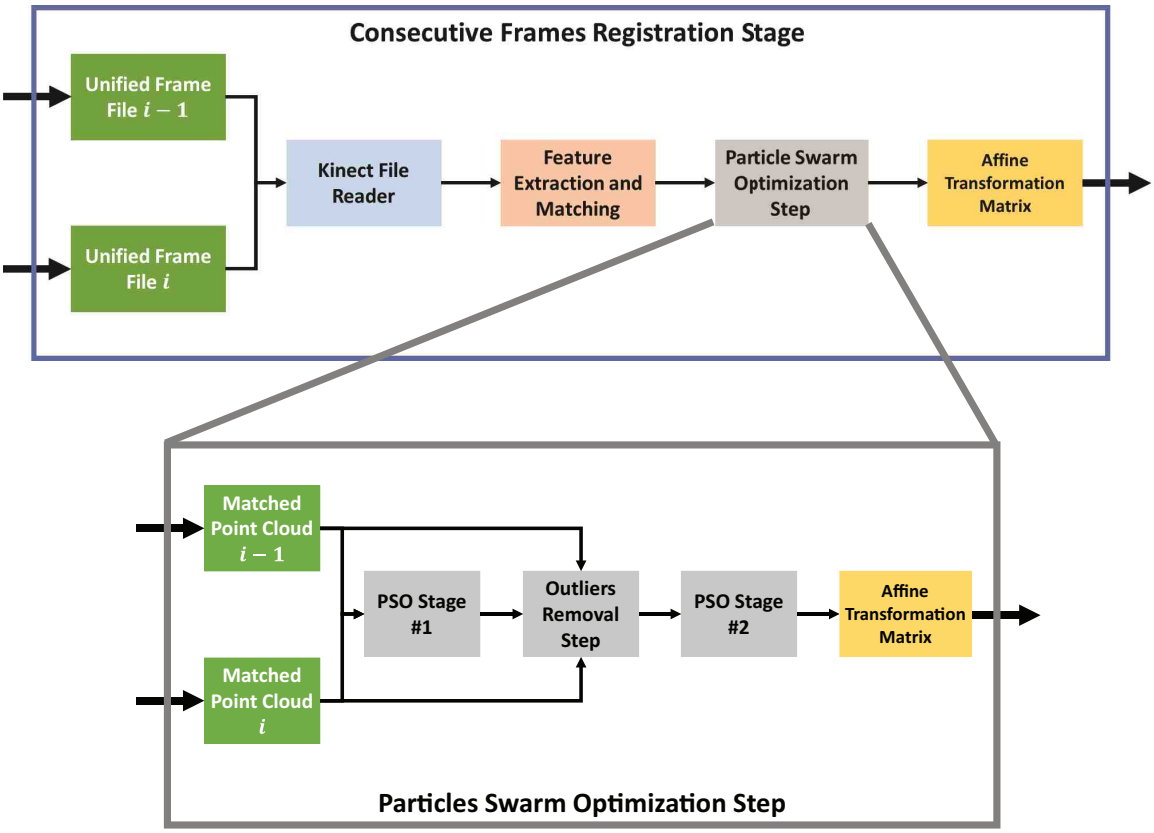


Figure 3.24. Particle Swarm Optimization Step.

Particle Swarm Optimization Step

The PSO step is used to obtain the 3D affine transformation matrices between consecutive point clouds. This step consist of an outlier removal step and two stages of PSO. The same PSO algorithm is used in both PSO stages. The outlier removal step is needed to eliminate outliers and static points which introduce artificial bias. The static points are matches generated by static objects in the scene.

The PSO stage inputs are the two matched point clouds and the output is the 3D affine transformation matrix. The outlier removal step takes the two point clouds (source and reference) and the 3D affine matrix and returns the two point clouds with the outlier entries removed. Figure 3.24 shows the block diagram of the proposed step.

In the previous outliers removal step, an outlier was any match whose distance is higher than the mean of all the match distances plus the standard deviation of all the match distances (Equation 3.51). The same concept is applied in the outlier removal step between the two PSO stages, except that the comparison uses the mean and the standard deviation of the distances between the corrected features.

In the case in which the features are perfectly aligned, the distance between the corrected input feature and its reference feature will be 0. On the other hand, if the distance between the corrected input feature and its reference feature increases relative to the distance between the uncorrected features, then it means that the matched feature is potentially an outlier because the input feature is moving away from the reference instead of moving closer to the reference. Therefore, in the outlier removal step, the distance between the corrected source point cloud and the reference point cloud are used to obtain the mean and the standard deviation used in Equation 3.51, instead of the distance between the source point cloud and the reference point cloud.

The implemented particle swarm optimization algorithm is as follows:

PSO(pointCloud1, pointCloud2)

1. **Initialize** parameters
2. **Compute** initial fitness value
3. **If** the initial fitness value is 0 **then** return identity matrix as the result
4. **Initialize** particles
5. **While** the stop criteria is not met
6. **Increase** the iteration counter
7. **For** each particle
8. **Find** the best local particle of the particles in the neighborhood
9. **For** each dimension of the particle
10. **Compute** the inertia weight for the dimension

11. **Compute** the velocity of the dimension
12. **Apply** the particle to pointCloud1
13. **Compute** the new fitness value
14. **Update** the particle's best local fitness value
15. **Update** the best global fitness value
16. **Check** if the stop criteria is met
17. **Return** the best global particle as the affine transformation matrix

Each particle represents a possible affine matrix that aligns both point clouds arrays, which means that each particle has 12 degrees of freedom. The fitness function chosen for this system is the sum of the distances between the corrected and reference features (See Equation 3.52). In the ideal case of a perfect match the fitness value will be equal to 0. Therefore, the smaller the value of the fitness function, the better the registration. In some rare occasions, the patient may remain immobile for some time. This implies that the respective frames will reflect no motion, which means that the affine transformation matrix between those frames is approximately an identity matrix. An initial fitness value is calculated to prevent the algorithm from running unnecessarily. If the initial fitness value is equal to 0, the algorithm considers that there was no movement between the two frames and returns an identity matrix as the result. The update of the position of each dimension of the particle is done using Equation 3.24. The velocity of each dimension of the particle is updated using Equation 3.36. Meanwhile, the inertia weight term is calculated using Equation 3.41.

$$fitness = \sum_{i=1}^{n} \sqrt{(x_{PC1,i} - x_{PC2,i})^2 + (y_{PC1,i} - y_{PC2,i})^2 + (z_{PC1,i} - z_{PC2,i})^2} \quad (3.52)$$

where, n is the number of matches, $PC1$ and $PC2$ are pointCloud1 and pointCloud2, respectively, and $x_{A,i}$, $y_{A,i}$ and $z_{A,i}$ are the x, y and z coordinates of the $i-$th feature in point cloud A, respectively.

An initialization step is used to generate the initial particles. Each particle is assigned an identification label, which is its location in an array. Then, the swarm is initialized using a completely random normal distribution, and a random value is assigned to each of the 12 degrees of freedom parameters of each particle. Also, the particles' velocity array, the best local fitness for each particle, the best local result for each particle and the best global particle are initialized using random values.

The acceleration coefficients c_1 and c_2 are set to 1.49 as suggested in [43]. To choose the swarm size, the neighborhood size, and the maximum number of iterations, several experiments were performed. A value for the maximum number of iterations is fixed to ensure that the process does not run indefinitely. Moreover, this also depends on the execution time of a single iteration, which is on average 0.807 ± 0.2118 ms. Assuming that the execution time per file should not exceed 30 seconds in the worst case scenario, then the maximum number of iterations will be $\frac{\text{execution time per file}}{\text{execution time for a single iteration}} = \frac{30}{0.807+0.2118} = 29447.87$, which is approximately 30000 iterations.

To choose the swarm size several experiments were carried out using different swarm and neighborhood sizes that were varied systematically while keeping the rest of the parameters constant. In particular, two neighborhood sizes were used: 15% of the swarm size and 30% of the swarm size. Figure 3.25 depicts the results of these tests were the swarm size was varied from 10 to 60 particles. From this figure, it can be concluded that a swarm size between 20 and 30 particles gives a good balance between runtime and number of iterations when neighborhood sizes are set to be 15% to 30% of the swarm size. Based on this, the chosen value for the swarm size for this project was set to 30.

Having chosen a swarm size of 30, a similar experiment was conducted to choose the neighborhood size. However, in this case only odd numbers where used for neighborhood sizes, as shown in Figure 3.27 Figure 3.26 shows the results obtained. For a swarm size of 30 particles, the best results where obtained with a neighborhood size, N_{size}, of 11 particles. Moreover, it can be concluded that for smaller neighborhood

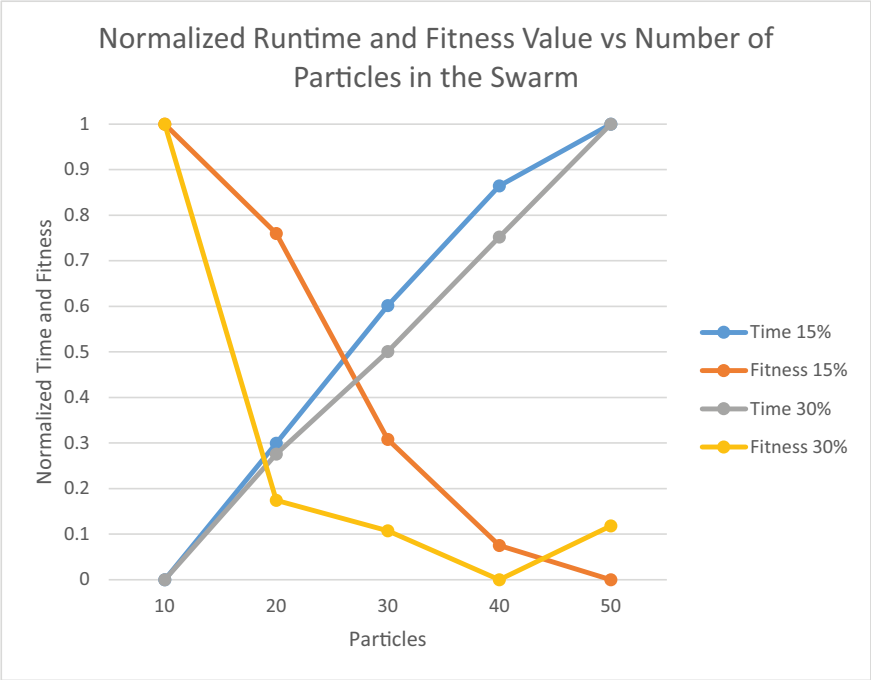


Figure 3.25. Comparison of the Normalized Runtime and Fitness Value vs the Number of Particles in the Swarm..

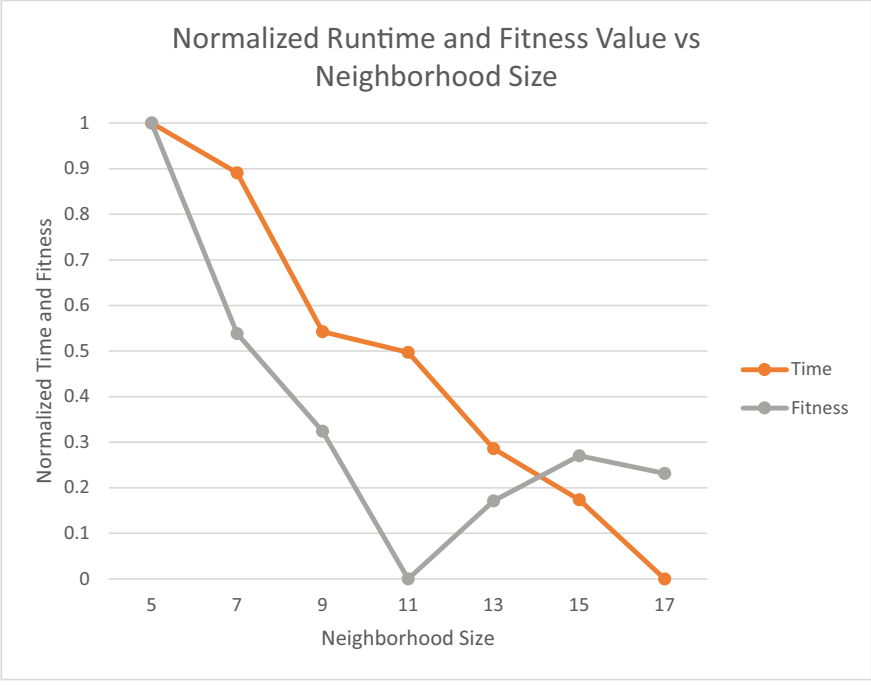


Figure 3.26. Comparison of the Normalized Runtime and Fitness Value vs the Neighborhood Size.

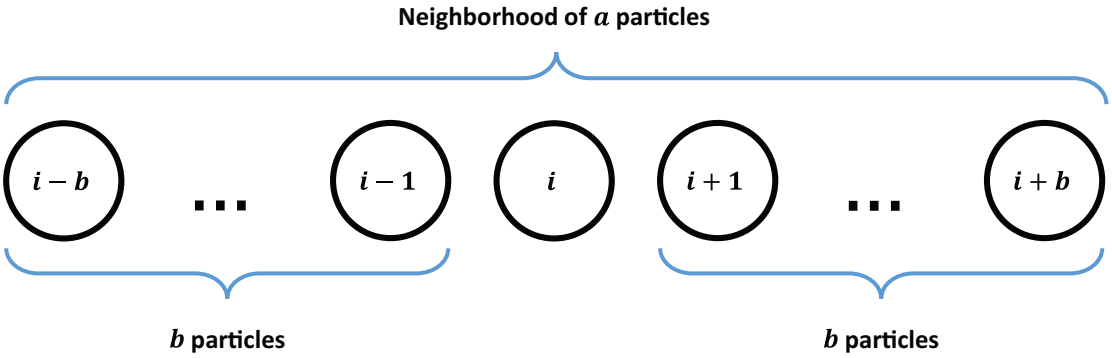


Figure 3.27. Neighborhood of size a for particle i.

sizes the algorithm will take more time to finish because the particles take longer to converge. On the other hand, larger neighborhood sizes take less time to finish because the particles take less time to converge, but the fitness value increases because the particles become more susceptible to getting stuck in a local minimum. Additionally, the local topology of the neighborhood implemented in this system was a wheel topology. During each iteration, each particle has its "own" neighborhood which is made up $\frac{N_{size}-1}{2}$ particles that precede it in the particles array and $\frac{N_{size}-1}{2}$ particles that come after it in the particles array. For example, for particle i, its neighborhood consists of all the particles whose indices range from $i - \frac{N_{size}-1}{2}$ to $i + \frac{N_{size}-1}{2}$. Each particle in the neighborhood communicates its results to particle i, which compares who has the best results based on the fitness value, but it does not communicates this to its neighbors. It uses the obtained best result as the $B_{best}(t)$ which it used to update its own position.

The algorithm uses one of two stopping conditions: the first condition is the maximum number of iterations and the second condition is that the difference between the last change and the average of the last 10 changes is less than 10^{-6}. Also, in each iteration all particles must update their positions before communicating their results to the swarm.

3.3.3 3D Model Registration Algorithm

In the previous stage, the Consecutive Frames Registration Stage, only the affine matrices that align consecutive frames are obtained. However, to perform motion correction, all frames must be aligned with respect to an initial reference frame. During the early phases of the development of the system, it was observed that to carry out the task of registration between the frames it has to be done sequentially. This has a major impact on the runtime of the algorithm. On the other hand, it was observed that the system's runtime could be improved by dividing the registration step in two stages: a stage in which the registration is performed over consecutive frames and a stage in charge of registering all the frames with the reference. The first stage only requires two consecutive frame files to perform registration; meanwhile, the second stage requires the affine matrices from the first stage in addition to the frame files.

Figure 3.28 describes the process by which the infrared images and point clouds have passed to this point through of the proposed system. The first step is to obtain the matching features between the infrared images I_{i-1} and I_i, then, the matches and the point clouds are then used to generate matched point clouds, which are used to perform the registration whose result is the affine matrix $T_{(i-1)i}$ which aligns point cloud i and point cloud $i-1$.

In the case where perfect registration is obtained, whenever the resulting affine matrix $T_{(i-1)i}$ is applied to point cloud i, it will generate point cloud $i-1$. Therefore, to obtain the affine matrix T_{0i}, which aligns point cloud i and point cloud 0 (the initial reference), it is enough to apply all the affine matrices sequentially starting with $T_{(i-1)i}$ and ending with T_{01}. Figure 3.29 illustrates this concept, where P_i is the original point cloud and F'_i is the registered version of P_i with respect to point cloud P_{i-1}. If the point clouds are perfectly aligned, F'_i and P_{i-1} will be equal, then, for example, to align P_2 with the reference, first apply T_{12} to P_2 which gives $F'_2 = P_1$, and then apply T_{01} to F'_2 which results in $F''_2 = F'_1 = P_0$.

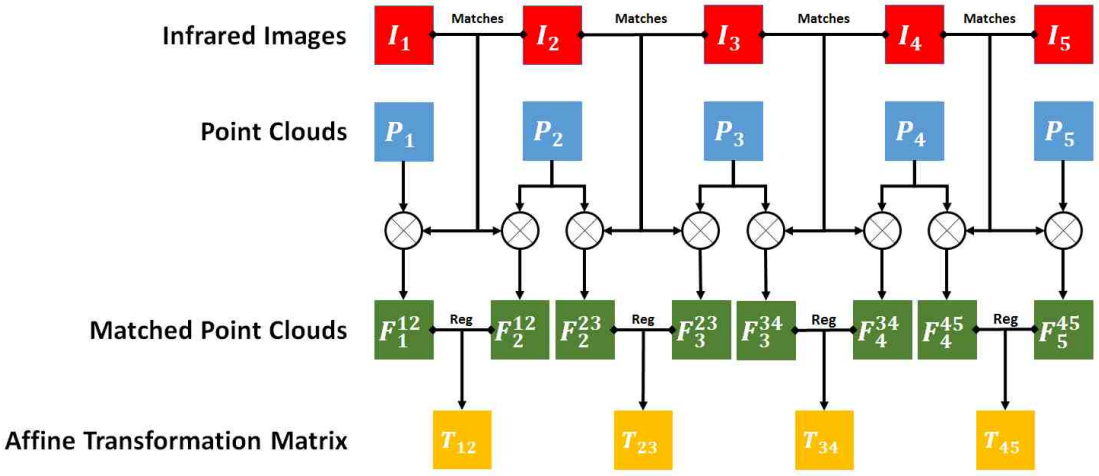


Figure 3.28. Infrared Images and Point Clouds Flow Chart.

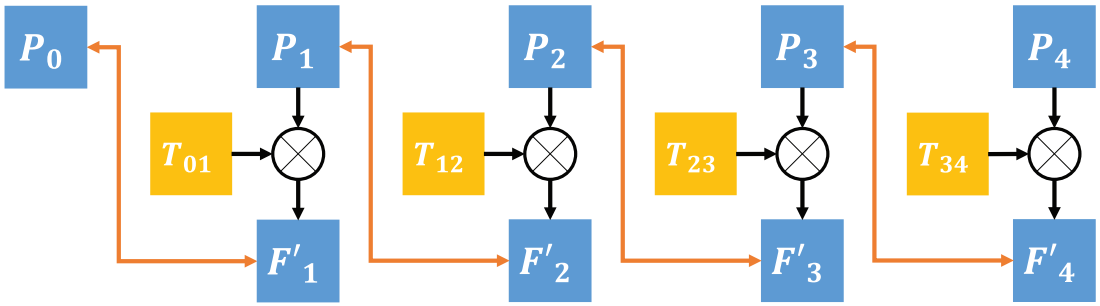


Figure 3.29. Ideal Registration Process.

In practice, no ideal results are obtained for the registration process. Even small differences in the transformation cause a registration error. With the procedure described above is applied, these errors accumulate along the frames causing a drifting behavior as more affine matrices are applied. To reduce the drifting effect due to the errors, a windowing method is used. In the proposed algorithm, the system will update the reference frame after k frames have been processed. This is depicted in Figure 3.30 which describes the workflow of the proposed algorithm for $k = 4$.

The first step of the algorithm is to perform the registration between frames i and $i - 1$. Once this task is done, the algorithm has to perform the registration between corrected frame i and frame $i - 2$ using all the i^{th} frames that have not been corrected with respect to the reference frame. This process is repeated until all the frames in the window of size k are registered with respect to the reference frame. Once all frames have been corrected, corrected frame k is set as the new reference for the next set of frames.

3.3.4 Motion Correction Step

The purpose of this stage is to perform motion correction on the DICOM images. This step utilizes, it takes the DICOM images from the PET/CT scanner and the affine transformation matrices obtained from the previous stage. The output generated is the motion corrected DICOM files. To choose the appropriate transformation matrix that has to be applied over the DICOM image, the time stamps of the DICOM file and the transformation matrix need to be aligned. Figure 3.31 depicts the workflow of this stage. The process is comprised of three steps: a pre-motion correction step where the DICOM files and the affine matrices are aligned and this results in a 3D volume that is reconstructed from the DICOM files, the motion correction step where the affine matrices are applied over their corresponding 3D volume, and the last step in which the motion corrected DICOM files are created.

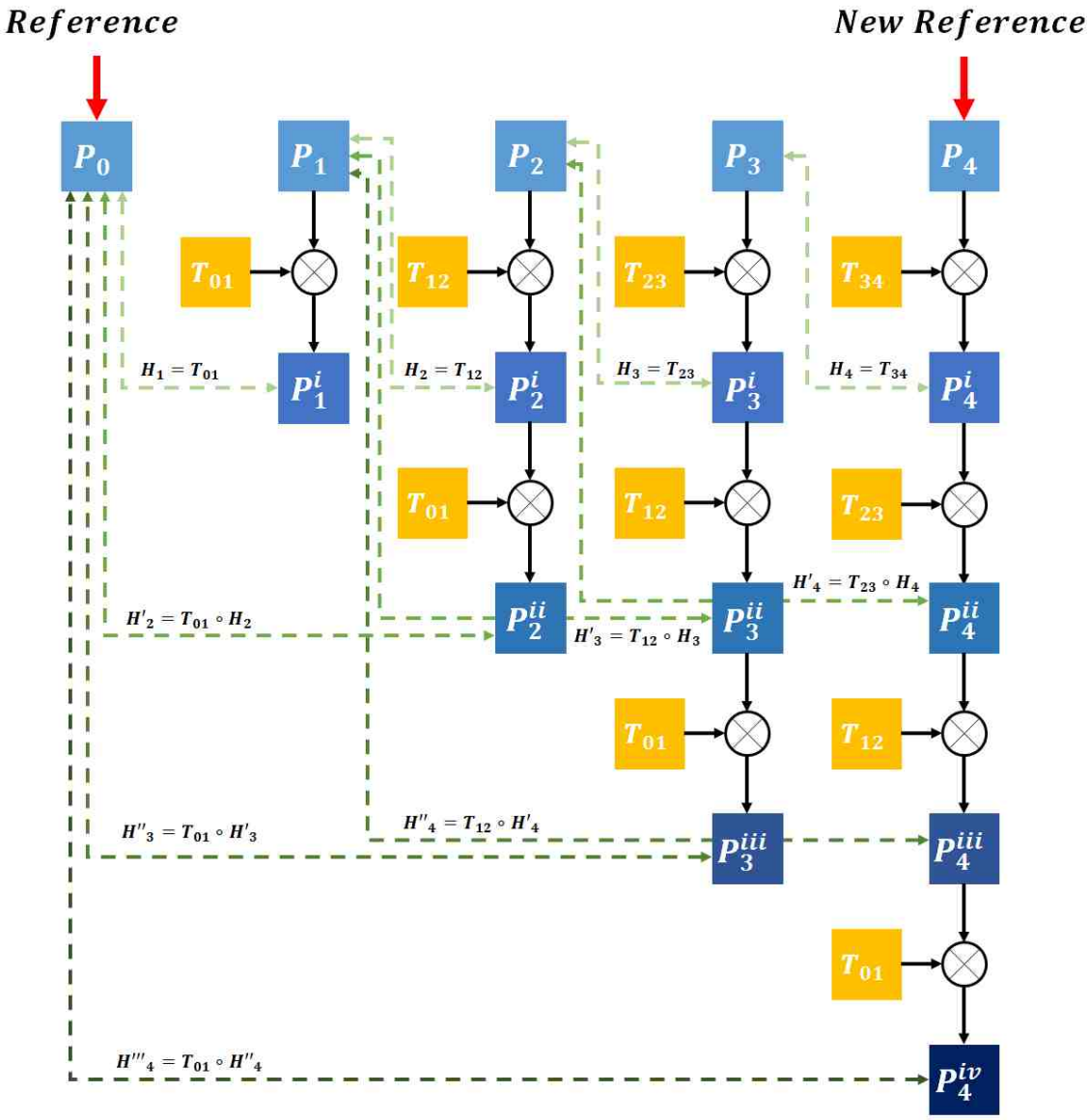


Figure 3.30. Windowing Algorithm Flow Chart.

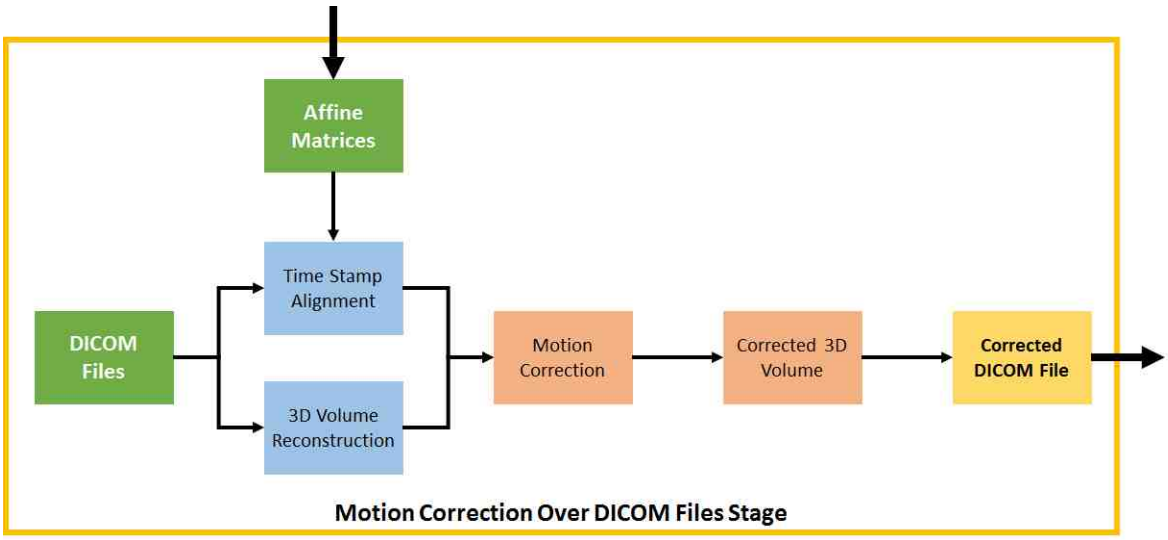


Figure 3.31. Motion Correction Stage.

Although the prior stage returns all the necessary affine matrices, this stage requires one to choose and apply only the appropriate matrix to the DICOM images. Usually, a certain number of DICOM files have the same acquisition time. In this case, the files will use the same affine matrix. To make the motion correction task efficient and since 3D affine matrices are obtained, a 3D volume will be constructed from the image slices contained in the DICOM files that share the same acquisition time. This is possible because the header of the DICOM file contains the following attributes: image position, image orientation, pixel spacing in the x-axis and y-axis, slice location, and slice thickness.

The image position attribute gives the x, y, and z coordinates of the upper left corner of the slice. The image orientation gives the direction cosines of the first row and the first column with respect to the patient. Image position and image orientation are used to properly order the slices in space. The pixel spacing attribute is the physical distance between the center of each 2D pixel in mm. It is specified by two values, where the first one is for the row spacing, $y_{spacing}$, and the second one is for the column spacing, $x_{spacing}$. Figure 3.32 depicts an example to illustrate this concept where each square represents a pixel and the orange squares represents the center of the pixel. The pixel spacing attribute allows the pixels of the slice to be spaced and positioned appropriately relative to a physical distance. The slice thickness attribute represents the width in mm of the PET scanner detector used to acquire the image, and the slice location is the relative position of the image plane expressed in mm. Using slice location attribute, the image slices can be placed in the proper location.

Each pixel in the image slice generates a point in the 3D model. Generating the 3D model, for each slice, it starts by placing the pixel in the upper left corner of the slice in the 3D space. This is iterated through each pixel in the slice placing them in the 3D space using the appropriate spacing, in the x and y axis, given by the pixel spacing attribute. The z coordinate of all the pixel for a specific slice is given by its slice location attribute, and the intensity of the point in 3D is given by its pixel value in the slice. This is illustrated in Figure 3.33.

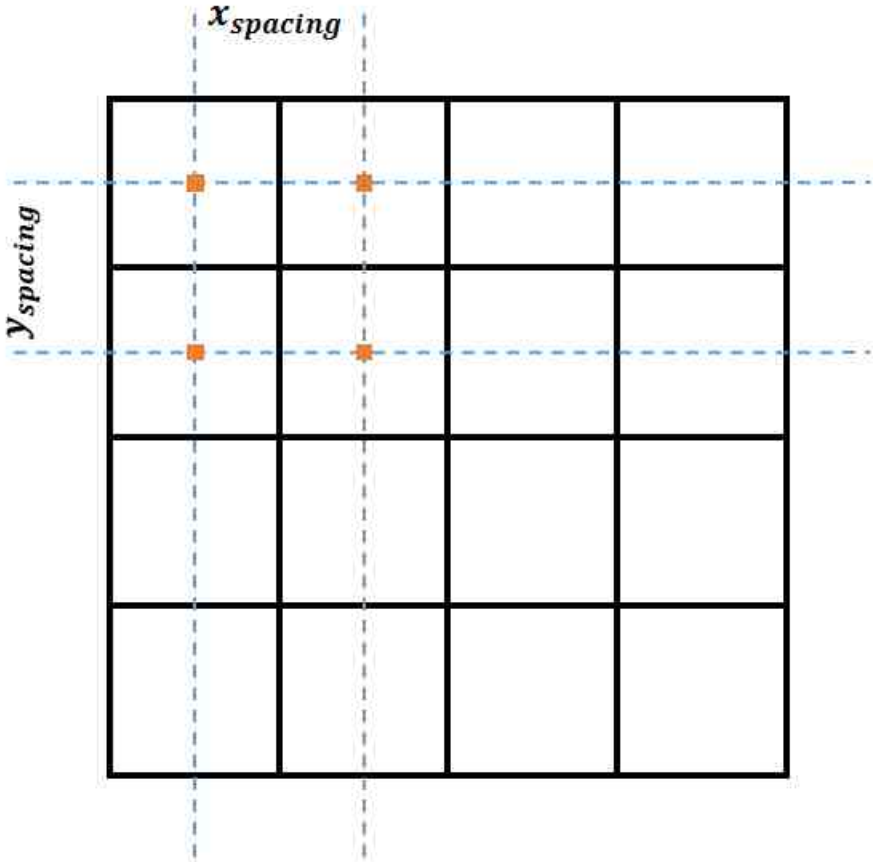


Figure 3.32. Pixel spacing example.

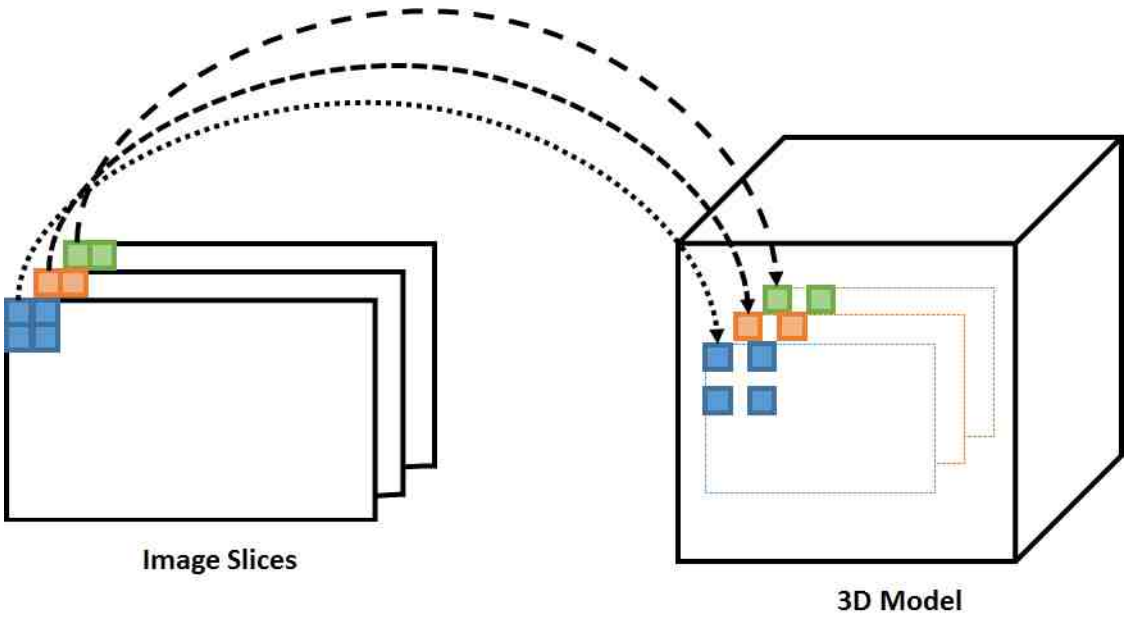


Figure 3.33. Process of generating a 3D model using the image slices.

Once the 3D volume is generated, the time stamp aligned affine matrix is applied to this volume. This process will return a motion corrected 3D model of the DICOM files for that specific acquisition time step. This corrected 3D model is then used to obtain the corrected 2D images which will be returned to the PET/CT scanner as a series of DICOM files. The number of 2D slices generated is equal to the number of slices given for the generation of the 3D model.

To generate the slices, the method places a 2D plane perpendicular to the z-axis of the 3D model at each slice location, then, it locates the upper left corner of the slice in the 2D plane and extracts the intensity value of that position and copies it into the corresponding pixel of the slice. To fill the rest of the pixels the technique "iterates" through the 2D plane, which is possible because the resolution of the output slice is the same as the resolution of the slice used to generate the 3D model and the pixel spacing is known. For example, to fill the first row of the slice, the 2D plane is sampled at every $k * x_{spacing}$, where k varies from 0 to the number of columns in the slice. Once all the pixels in the slice are obtained, the slice image is stored in the output DICOM file.

3.4 Graphical User Interface

The proposed motion correction algorithm is controlled using a graphical user interface. The graphical user interface consists of four tabs, each of controls a different aspect of the implemented system. The graphical user interface tabs are:

- Camera Tab

  The Camera tab is used to control the capture of the infrared images and the depth maps during the PET/CT examination, in addition to allowing the user to choose the region of interest. Figure 3.34 shows the initial Camera Window. The Camera Window displays the infrared images obtained by the Kinect in real-time. The region of interest is specified using a user specified rectangle. This rectangle can be resized and moved within the margins of the image while

Figure 3.34. Initial view of the camera tab.

the Camera is not capturing images (either, before the algorithm starts image acquisition or while the acquisition is paused). To resize the rectangle, the user needs to use any of the 8 square grips placed on the rectangle. To move the rectangle, use a 'drag and drop' operation. Before starting a capture, a subject or study ID must be entered. This can be done using the 'Enter Subject ID' button. Once clicked, a dialog box will appear (as shown in Figure 3.35). This button will not appear again until acquisition is finished. Also, before capturing images for a new study, the user needs to finish the prior capture. Figure 3.38 depicts the screen that allows the user to finish the acquisition process. This view is the same while the image acquisition is paused. Figure 3.36 shows the screen view of the system when it is ready to start the image acquisition process and Figure 3.37 shows the screen view during the image acquisition process.

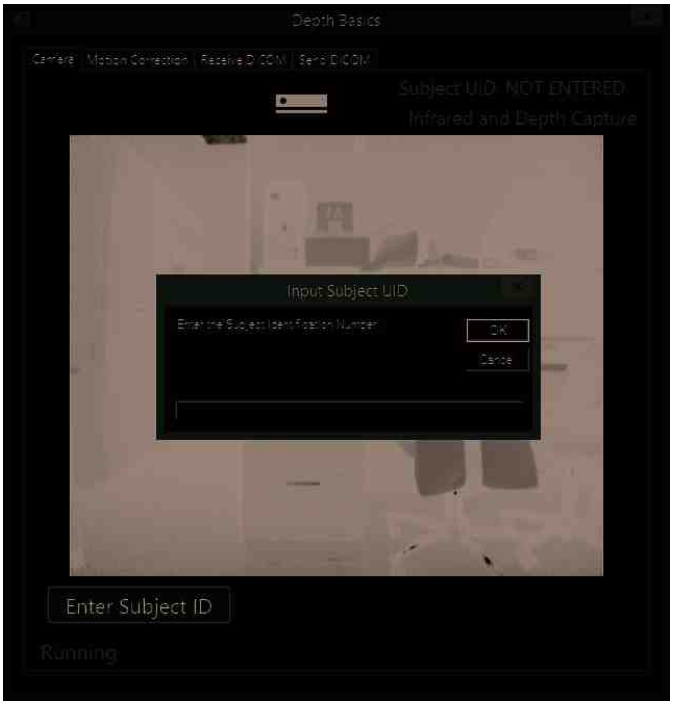


Figure 3.35. View of the 'Enter Study ID' dialog box.

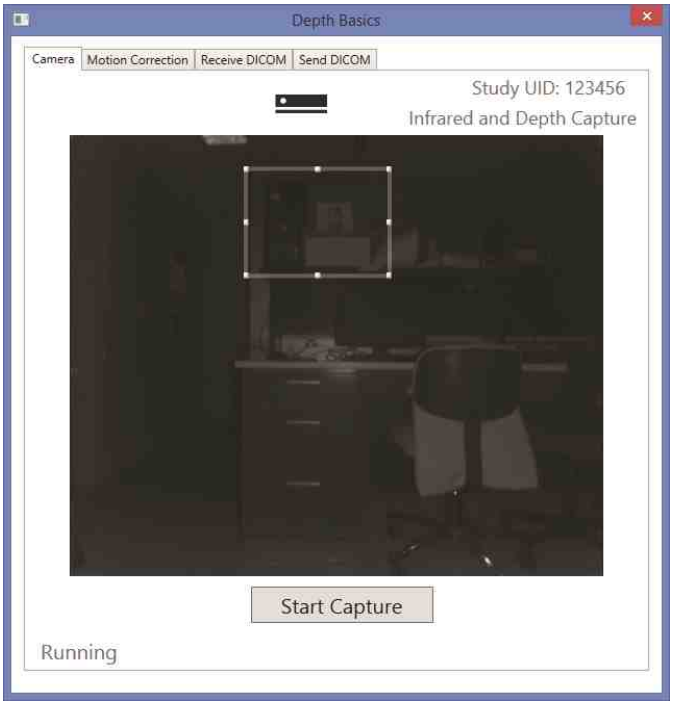


Figure 3.36. View of the tab when the system is ready to start the image acquisition.

Figure 3.37. View of the tab during the images acquisition task.

Figure 3.38. View of the tab while the image acquisition is paused.

- Motion Correction Tab

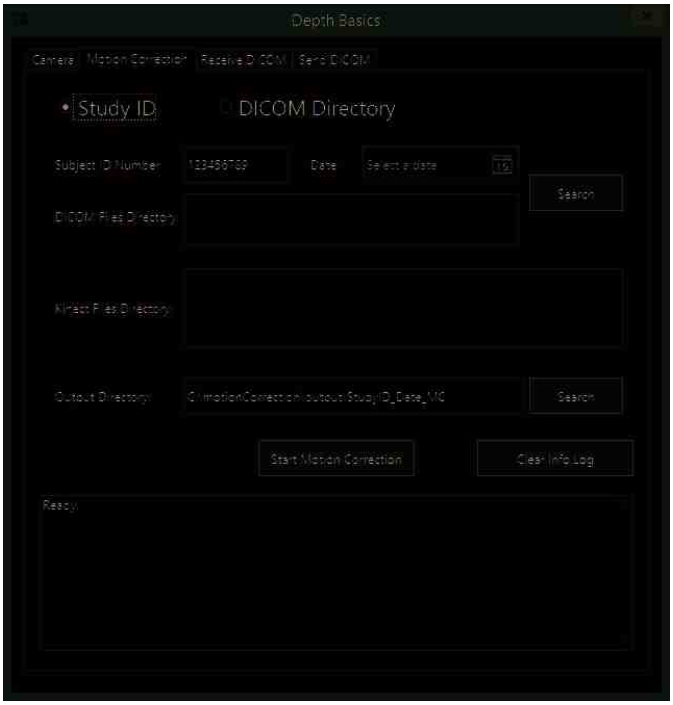
  The Motion Correction tab is used to apply the motion correction on the DICOM files. Before starting the motion correction, it is expected that the DICOM files and the Kinect files have been stored on the computer. The DICOM files can be selected in two ways: by specifying the Study ID number and the date of the study, as shown in Figure 3.39(a), or by specifying the directory containing the DICOM files to be corrected, as depicted in Figure 3.39(b). Once the DICOM files have been selected, the user has to select the Kinect Files directory from a list of possible directories. If the Kinect directory list remains empty after choosing the DICOM files directory, it means that no Kinect directory exists for the chosen DICOM directory. The user can specify an output directory. If the text box is left empty, the system will generate a new directory for the corrected files at a default location. Once these three parameters are specified, the motion correction algorithm can be started. The progress and any possible error generated during the execution of the algorithm will show up in the Info Log located at the bottom of the window.

- DICOM Receiver Tab

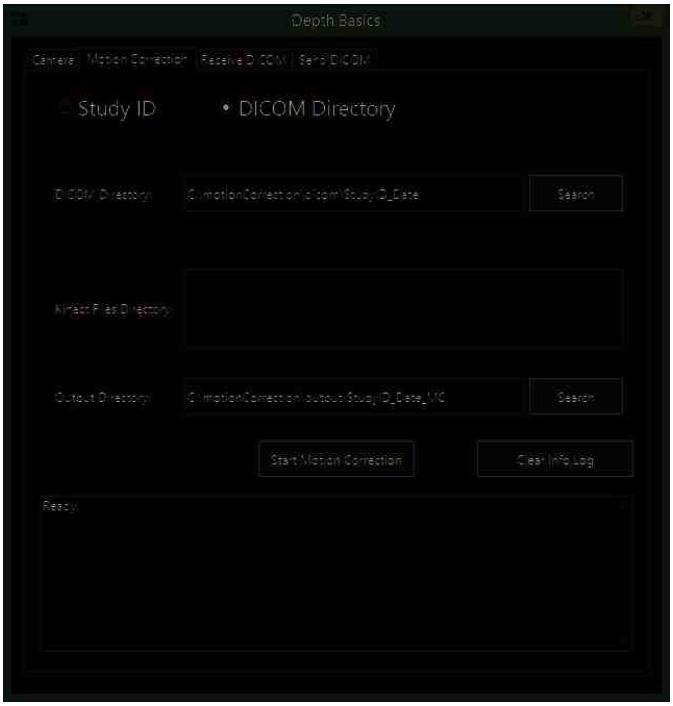
  The DICOM Receiver tab is used to enable a receiving server that allows the computer to receive DICOM files from the PET/CT scanner. The server requires two parameters: the directory in which the received DICOM files will be stored and the Port number. If the directory is not specified, the files will be saved in a default location. Figure 3.40 displays the DICOM Receiver window.

- DICOM Sender Tab

  The DICOM Sender tab is used to send DICOM files towards the PET/CT scanner or a specified machine. To send the DICOM files, three parameters are required: the directory that contains the DICOM files, the Port number and the IP address of the endpoint. Figure 3.41 depicts the DICOM Sender window.



(a) Study ID and Date



(b) DICOM Files Directory

Figure 3.39. Motion Correction Views

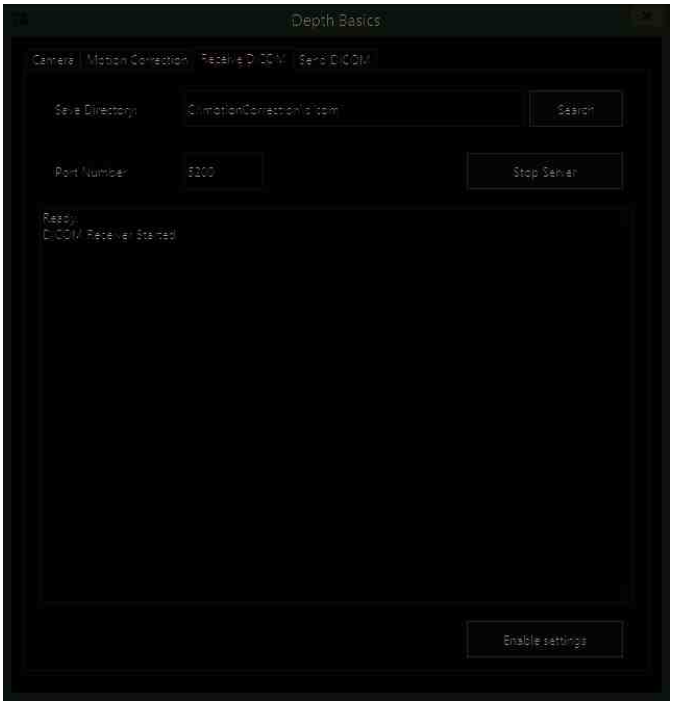


Figure 3.40. DICOM Receiver View

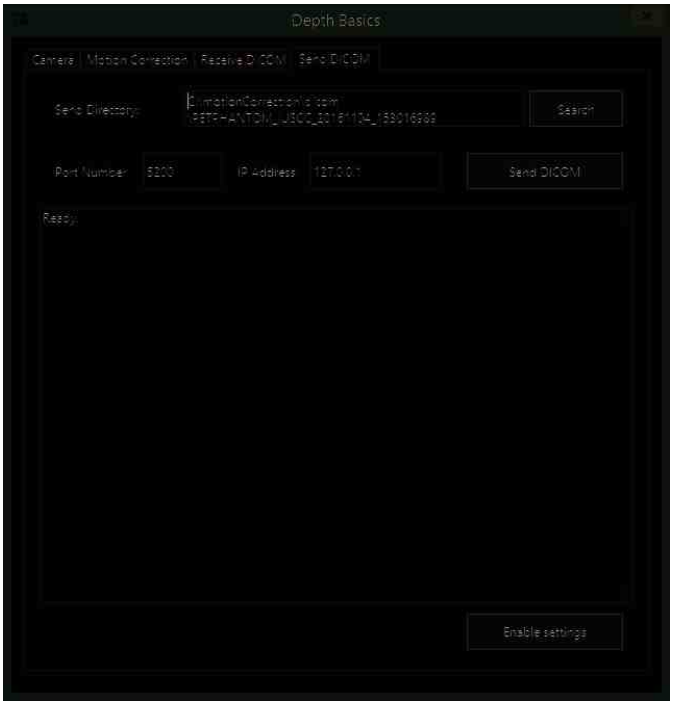


Figure 3.41. DICOM Sender View

4. RESULTS AND DISCUSSION

This chapter presents the results obtained from the implementation of the registration algorithm. To obtain these results, the proposed Microsoft Kinect Stage, described in Section 3.2.1, was implemented in the PET/CT scanner and images were acquired with a subject inside the scanner. The distance between the subject and the Kinect sensor was 0.9144 meters. No PET/CT data was acquired at this time. The acquired images were used as input to the Consecutive Frames Registration, described in Section 3.2.2, and the 3D Model Registration Algorithm, described in Section 3.2.3.

The different stages of the proposed system were tested separately to permit optimization of each component (e.g. unit test). The use of the transformation matrix over the medical images files was tested on various images and it was found to be reliable. The current tests are directed at ensuring that the integration between the Microsoft Kinect Stage, the Consecutive Frames Registration Stage and 3D Model Registration Stage work as expected.

During the initialization tests (See Figures 3.25 and 3.26) it is observed that the swarm size and the neighborhood size had a great influence on the runtime of the algorithm and the final distance between matched features. As the number of particles in the swarm increases, the runtime increases because more particles have to be processed but the fitness value decreases because there are more particles searching for the optimal solution in the solution space. As the neighborhood size increases the running time and the fitness value decreases, but the fitness value decreases to a limit and then begins to increase again. This behavior occurs because for smaller neighborhoods, the particles require more iterations to converge to an optimal solution. On the other hand, for large neighborhoods, more particles communicate with each other which makes the particles to converge faster, but this point of convergence can be the local minimum of one of the particles instead of a global minimum.

The raw infrared images and depth maps obtained from the Microsoft Kinect are shown in Figure 4.1. The parameters of the region of interest are: the ROI's upper left corner is located at position $x = 177$ and $y = 119$, its height is 118 pixels and it's width is 152 pixels. As can be seen in the infrared images of Figure 4.1, it is not possible to make the patient's head occupy the entire size of the infrared frame. Scene background data (such as: the PET/CT scanner gantry, interior and, bed) is captured along the patient's head and part of their body. Therefore, a region of interest is used to delimit the area where the patient's head will be confined. This facilitates the extraction of the object of interest (in this case, the patient's head) from the background of the scene. Each Kinect frame file has a size of 4370 KB. During the acquisition of these images, the lightning conditions in the room remained constant and the lights were at a subdued level.

Figure 4.2 shows the results generated using two consecutive frames. The point clouds are generated using the depth maps and the region of interests. The infrared images and the region of interest are used to obtain the infrared feature matches. The quiver plots are the 3D motion vectors between the extracted and matched features of the two point clouds. For each arrow, the head of the arrow is the location of the source feature (feature in the right point cloud) and the tail of the arrow represents the location of the matched reference feature (corresponding matched feature in the left point cloud).

Figures 4.3, 4.4, 4.5 and 4.6 show the results using seven frames which are equivalent to 3 seconds of motion. The proposed system average runtime using these seven frames as input is 32.732 ± 0.393 seconds. The average number of infrared matches obtained for these seven frames is 32. In the case where the registration is able to perfectly align the two point clouds, the length of each arrow in the quiver plots of the corrected matches would be equal to 0. Due to several factors, such as noise generated by the environment, and in the Kinect sensor, the registration is far from perfect and will not return a distance value of 0 for any match. On average, the length of all the

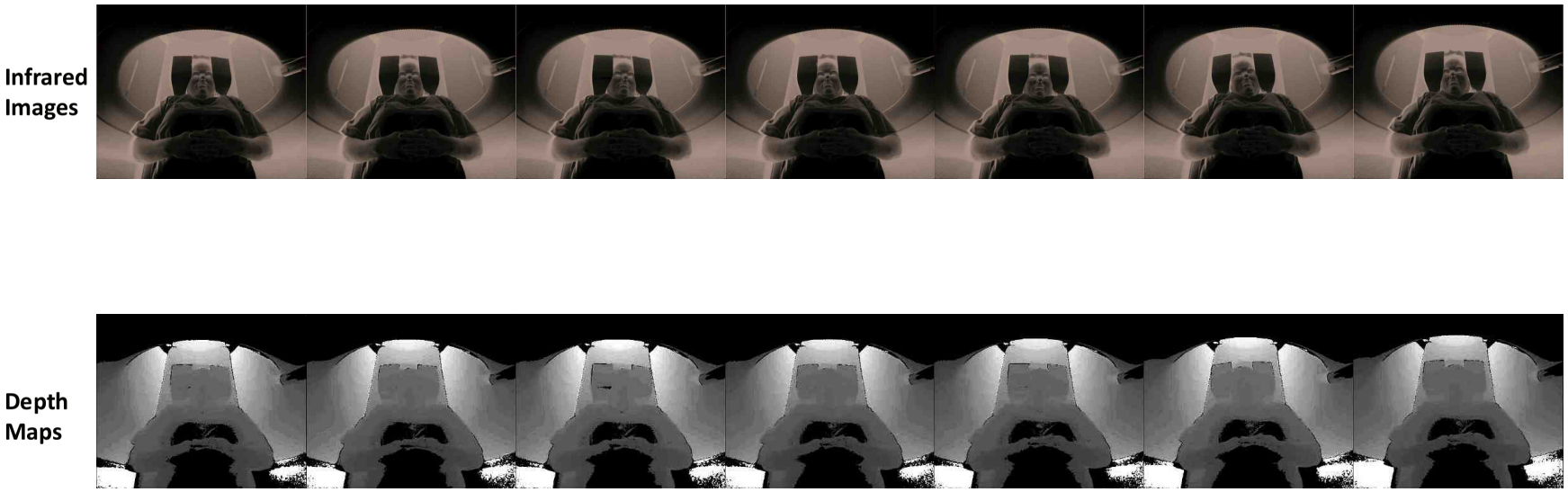


Figure 4.1. Raw IR Images and Depth Maps Acquired using the Microsoft Kinect.

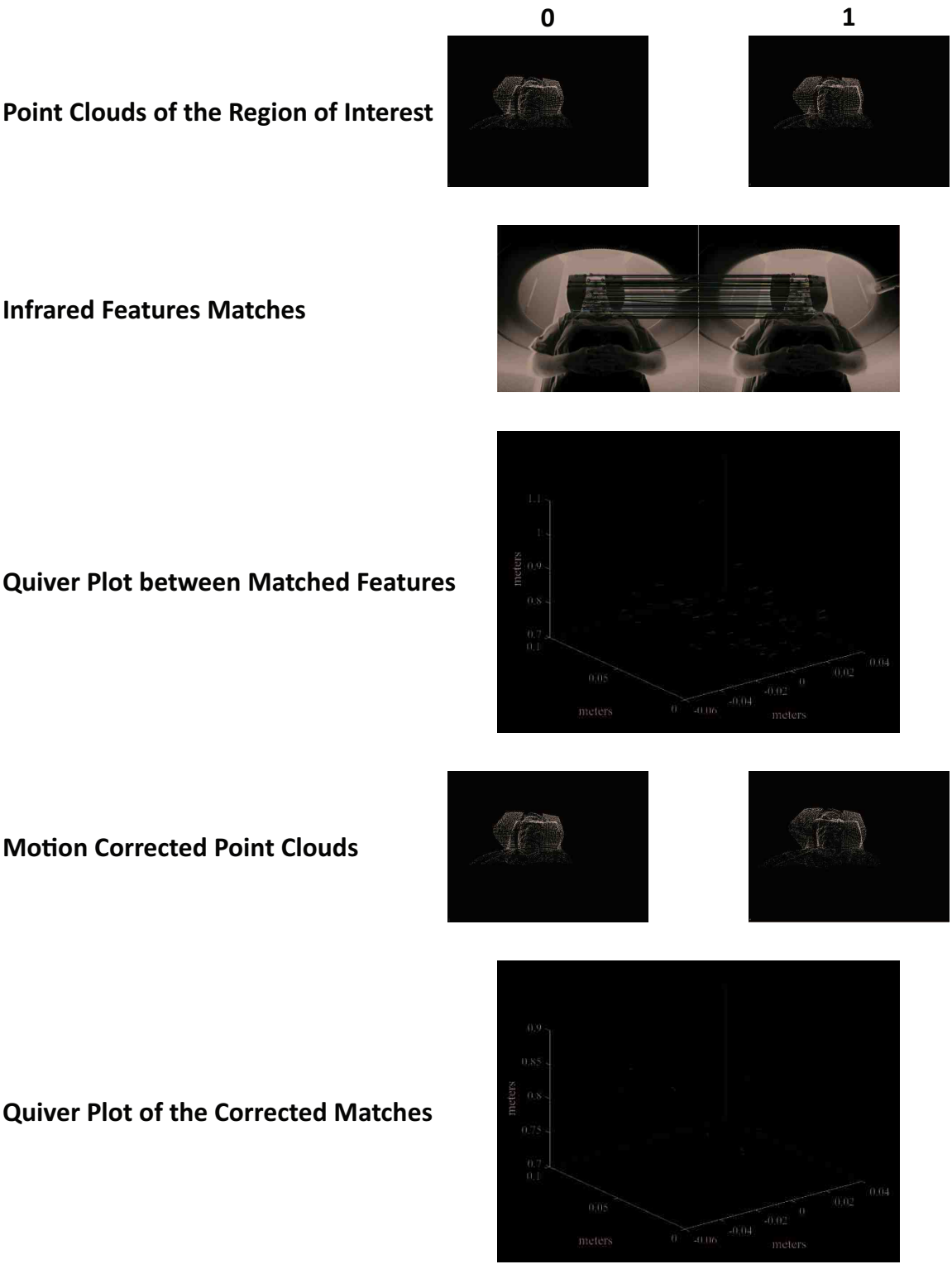


Figure 4.2. Results obtained from two consecutive frames.

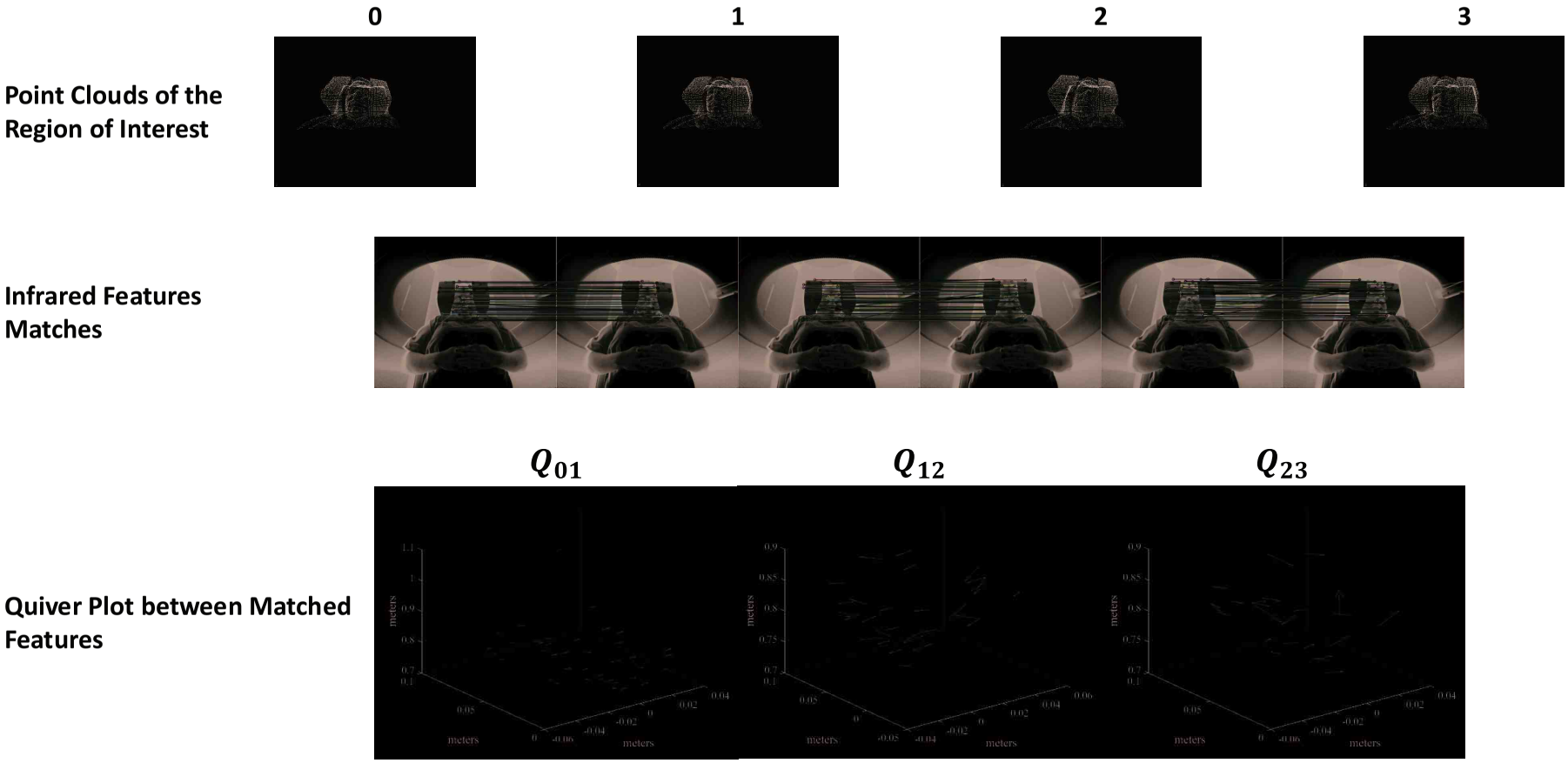


Figure 4.3. Original Point Clouds, Infrared Matches and Quiver Plots of the Matches for Frames 0 to 3.

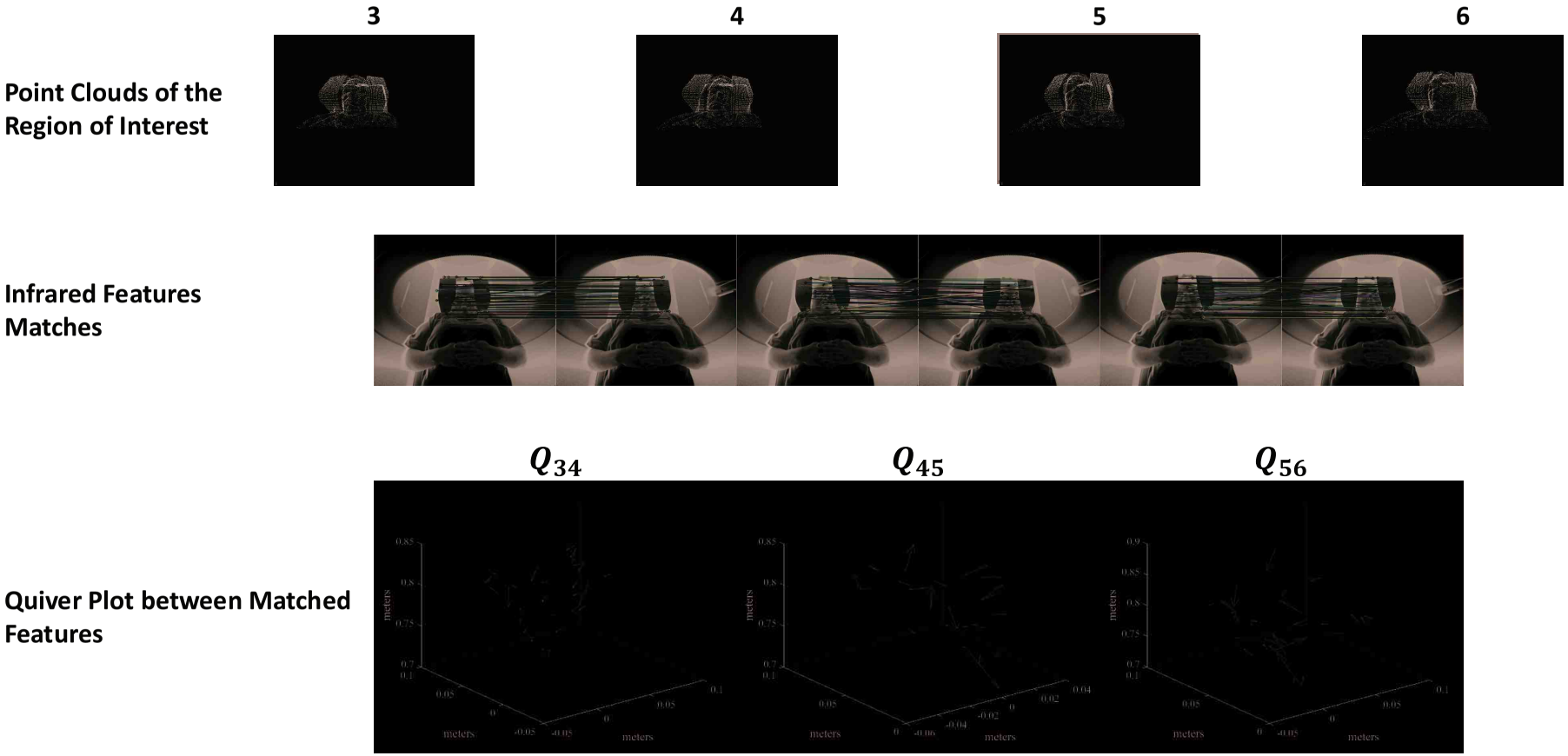


Figure 4.4. Original Point Clouds, Infrared Matches and Quiver Plots of the Matches for Frames 3 to 6.

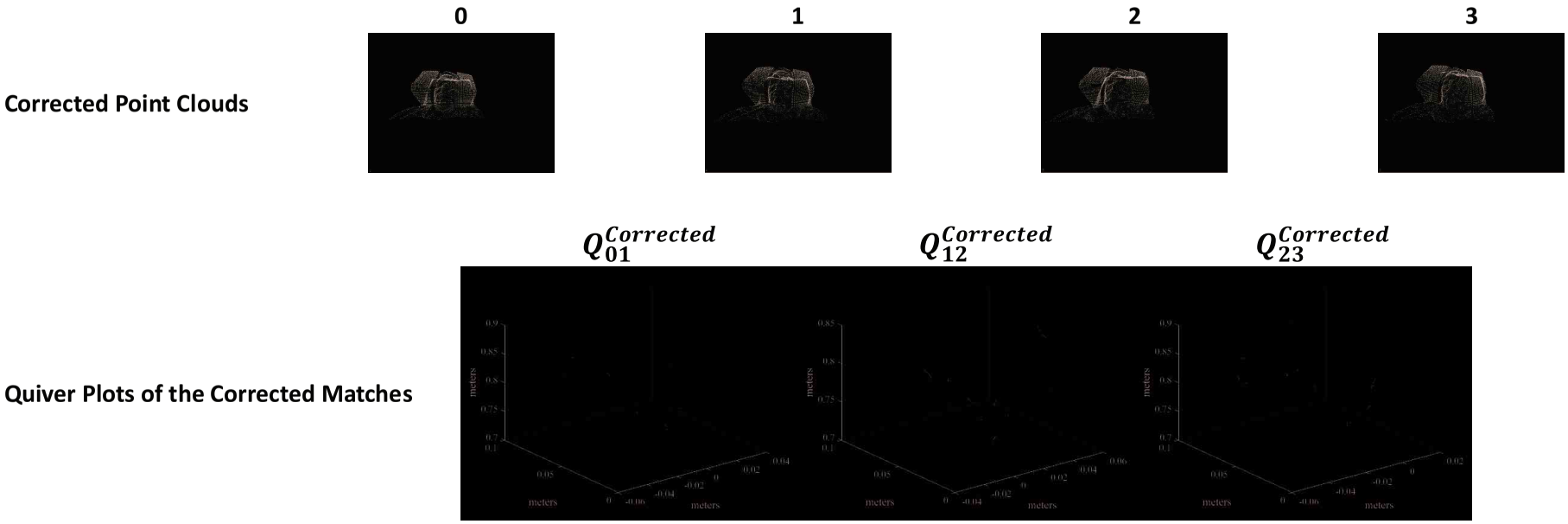


Figure 4.5. Corrected Point Clouds, Quiver Plots of the Corrected Matches for Frames 0 to 3.

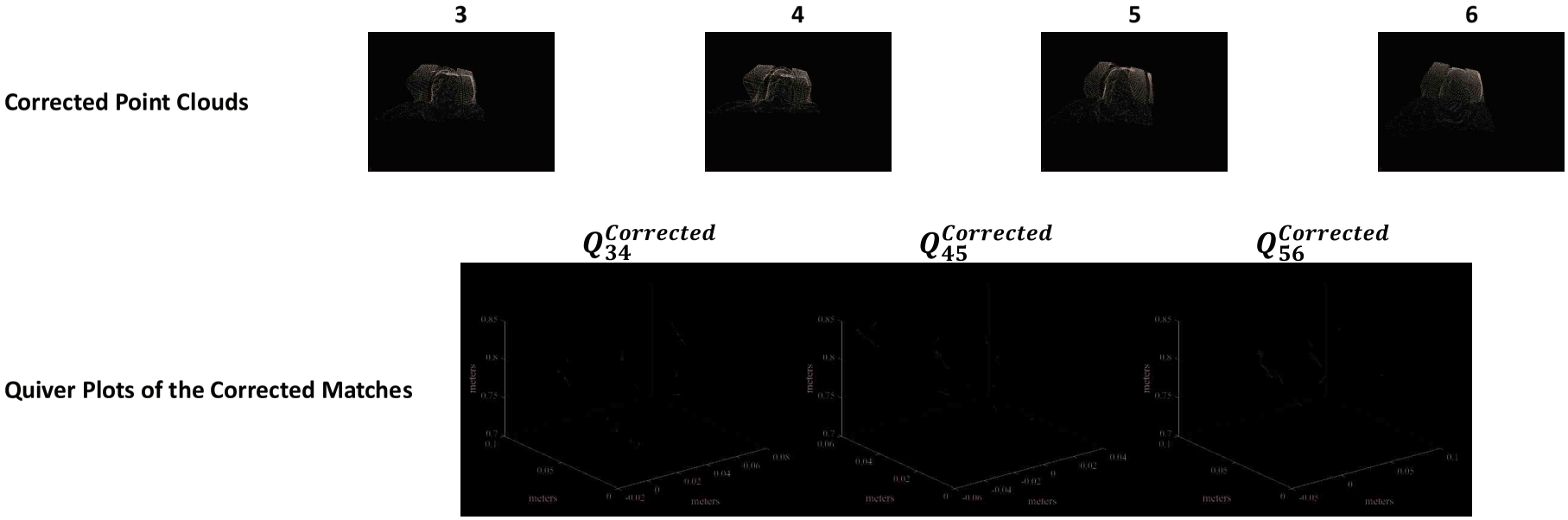


Figure 4.6. Corrected Point Clouds, Quiver Plots of the Corrected Matches for Frames 3 to 6.

arrows in the quiver plots between matched features is: 0.01169 ± 0.00725 meters. By comparison, the average length of all the arrows in the quiver plots of corrected matches is: 0.00379 ± 0.00417 meters. This indicates on average, the distances between the reference features and the source features was reduced by a 67%. Table 4.1 shows the average length of the arrows for the quiver plots between matched features and Table 4.2 shows the average length of the arrows for the quiver plots of the corrected matches. All distances in the tables are in meters.

In the literature, there are two types of motion correction systems: marker-based systems such as Polaris Vicra, and marker-less systems [14] [16] [9] [8]. A drawback of marker-based tracking systems is that the markers are likely to move during the scanning process leading to incorrect motion compensation, and may create discomfort for patients. The motion correction system proposed in this thesis is a marker-less system. In addition, it is not limited to the tracking of a specific object or part of the body such as tracking the chest motion [5], or as Meyer et.al. [14], where they try to estimate the position and orientation of the head of a person. Like the marker-less tracking systems mentioned earlier [14] [16], the proposed system in this thesis uses the depth information obtained from the Microsoft Kinect, but it uses the infrared information instead of the color information. This change was necessary because during a PET/CT scan the lighting conditions in the room can shift to the point where the lights are turned off completely for the patient's comfort or experimental procedures. The tracking systems presented in Sections 2.4.3 and 2.5 use face detection algorithms (e.g. Viola-Jones) [14] or perform the detection using depth edges to obtain the tracked object [16]. In contrast, the system proposed in this work uses a feature extraction and matching (SURF combined with FLANN) approach.

The Kinect SDK possess face tracking tools, but these require the Microsoft Kinect sensor to be placed in a specific orientation because the system uses the Kinect incorporated accelerometer. In addition, the Kinect SDK face tracking algorithm requires the face to be well lit without any harsh shadows on it. The implemented

Table 4.1.
Length of the arrows in the quiver plots between the matched features.

Frames	Average (m)	Standard Deviation (m)	Min (m)	Max (m)
Frame 0 & Frame 1	0.00540	0.00147	0.00304	0.00883
Frame 1 & Frame 2	0.01217	0.00407	0.00496	0.02204
Frame 2 & Frame 3	0.01646	0.00603	0.00917	0.03345
Frame 3 & Frame 4	0.00971	0.00417	0.00332	0.02125
Frame 4 & Frame 5	0.01774	0.00821	0.00993	0.04964
Frame 5 & Frame 6	0.01280	0.00935	0.00400	0.03716

Table 4.2.
Length of the arrows in the quiver plots of the corrected matches.

Frames	Average (m)	Standard Deviation (m)	Min (m)	Max (m)
Frame 0 & Frame 1	0.00148	0.00088	0.00015203	0.00307
Frame 1 & Frame 2	0.00313	0.00263	0.00000027	0.00765
Frame 2 & Frame 3	0.00148	0.00088	0.00015203	0.00307
Frame 3 & Frame 4	0.00511	0.00286	0.00000016	0.00969
Frame 4 & Frame 5	0.00509	0.00489	0.00000096	0.01782
Frame 5 & Frame 6	0.00820	0.00638	0.00008816	0.01768

system in this project does not required the Kinect sensor to be positioned in a specific orientation to work and is able to obtain the motion parameters from people lying down as well as it is able to work in poor lighting conditions.

5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

This thesis describes a motion correction system for medical files. The system consists of an algorithm capable of storing, acquiring and tracking movements of a patient within a PET/CT scanner using the Microsoft Kinect, an algorithm used to perform motion correction in medical files based on rigid body registration, and a GUI used to receive and transmit medical files. The proposed system is invariant to the lighting conditions of the room and does not require the use of markers to obtain the patient motion information. In addition, it does not require prior initialization except for delimiting the region of interest in acquired frames.

Some disadvantages of the proposed system are its dependency on features extraction and matching step which can affect the runtime and accuracy of the algorithm. Although the motion of the patient was considered an affine rigid transformation, the actual motion information obtained may not be non-rigid.

The proposed system uses PSO to find the parameters needed to perform rigid registration. In early tests of the algorithm, it was noticed that the PSO stage is susceptible to outliers. Thus two stages were needed for outlier removal.

The advantages of the proposed system is that motion correction in PET/CT scans can help physicians to observe a scanned area in a more detailed and precise manner. For example, it will help determine precisely where a specific structure is located (e.g. a carcinoma, a lymph node). It can also prevent misdiagnoses generated by motion blurred scans. Moreover the use of motion correction systems in patient care may reduce the need to expose the patient to ionizing radiation because of the need to re-image the patient due to the poor quality obtained in the PET/CT images. Ionizing radiations does not produce harmful effects on human's health if

used in small quantities, but after certain thresholds, it can affect the function of tissues and/or organs, in addition to side effects such as hair loss, radiation burns, and acute radiation syndrome [49].

5.2 Future work

During the development of this system, motion was approximated as a rigid affine transformation. An important improvement for the system would be to consider the motion as a non-rigid transformation. Rigid transformations serve to obtain a good estimate of the transformations generated by a patient's movements. But during a PET/CT scan, patient movement may be actually non-rigid, due to different factors such as tissue movements due to breathing which are non-rigid. Also, features matching was performed using Nearest Neighbors Search, the use of another feature matching algorithm, such as Iterative Closest Point, should be evaluated.

Due to limitations in the placement of the Kinect sensor, the proposed system was limited to correcting the movements of the patient's head. An idea to improve the system is to make it capable of correcting the movements of a patient's entire body. The acquisition frame rate for this project is not variable throughout the PET/CT examination; however, the use of an adaptive acquisition frame rate should be evaluated to see if it can improve the motion tracking. Also, the use of GPU based computing should be evaluated to improve the running time of the algorithm. Furthermore, the use of the GPU could allow the system to perform the motion correction task in near real-time.

In the literature algorithms that are capable of tracking objects using only 3D information captured by a sensor have been reported in [18] [19] [50]. The implementation of one of these algorithms would significantly improve the proposed system due to the elimination of its dependence on the use of the infrared information to perform the registration.

Image matching and registration in the projection domain has been studied previously [51] [52] [53]. Motion correction in the proposed system was performed in the image domain. A possibility for future research is to determine whether the motion correction and tracking can be performed in the projection domain. If a system capable of making the motion correction in the projection domain is developed, it would be possible to make the corresponding corrections before the reconstruction of the medical images.

Lastly, the proposed system was focused on motion correction in PET/CT scans, but there are other tools used to diagnose health problems in patients which suffer from similar issues (e.g. ultrasound and magnetic resonance imaging). The use and implementation of the proposed system for these tools should be evaluated.

REFERENCES

REFERENCES

[1] E. Lin and A. Alavi, *PET and PET/CT: A Clinical Guide*, second edition ed. Thieme, 2009.

[2] D. C. Owens, E. C. Johnstone, and C. Frith, "Spontaneous involuntary disorders of movement: Their prevalence, severity, and distribution in chronic schizophrenics with and without treatment with neuroleptics," *Archives of General Psychiatry*, vol. 39, no. 4, pp. 452–461, 1982.

[3] J. M. Kane, P. Weinhold, B. Kinon, J. Wegner, and M. Leader, "Prevalence of abnormal involuntary movements (spontaneous dyskinesias) in the normal elderly," *Psychopharmacology*, vol. 77, no. 2, pp. 105–108, 1982.

[4] R. Menezes, A. Pantelyat, I. Izbudak, and J. Birnbaum, "Movement and other neurodegenerative syndromes in patients with systemic rheumatic diseases: A case series of 8 patients and review of the literature," *Medicine*, vol. 94, no. 31, 2015.

[5] P. J. Noonan, J. Howard, D. Tout, I. Armstrong, H. A. Williams, T. F. Cootes, W. A. Hallett, and R. Hinz, "Accurate markerless respiratory tracking for gated whole body pet using the microsoft kinect," *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 3973–3974, October 2012.

[6] X. Jin, T. Mulnix, B. Planeta-Wilson, J.-D. Gallezot, and R. E. Carson, "Accuracy of head motion compensation for the hrrt: Comparison of methods," *2009 IEEE Nuclear Science Symposium Conference Record*, pp. 3199–3202, October 2009.

[7] X. Jin, T. Mulnix, J.-D. Gallezot, and R. E. Carson, "Evaluation of motion correction methods in human brain pet imaging a simulation study based on human motion data," *Medical Physics*, vol. 40, no. 10, September 2013.

[8] O. V. Olesen, J. M. Sullivan, T. Mulnix, R. R. Paulsen, L. Hojgaard, B. Roed, R. E. Carson, E. D. Morris, and R. Larsen, "List-mode pet motion correction using markerless head tracking: Proof-of-concept with scans of human subject," *IEEE Transactions on Medical Imaging*, vol. 32, no. 2, pp. 200–209, February 2013.

[9] P. J. Noonan, J. Howard, T. F. Cootes, W. A. Hallett, and R. Hinz, "Realtime markerless rigid body head motion tracking using the microsoft kinect," *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 2241–2246, October 2012.

[10] Microsoft, *Skeletal Tracking*, [Online; Last Date Accessed: March 14, 2017], https://msdn.microsoft.com/en-us/library/hh973074.aspx.

[11] Y. Li, L. Berkowitz, G. Noskin, and S. Mehrotra, "Detection of patient's bed statuses in 3d using a microsoft kinect," *36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5900–5903, August 2014.

[12] P. Kochunov, J. L. Lancaster, D. C. Glahn, D. Purdy, A. R. Laird, F. Gao, and P. Fox, "Retrospective motion correction protocol for high-resolution anatomical mri," *Human Brain Mapping*, vol. 27, no. 12, pp. 957–962, 2006.

[13] C. Studholme, D. J. Hawkes, and D. L. Hill, "Normalized entropy measure for multimodality image alignment," *Proceedings of the SPIE Conference on Image Processing*, vol. 3338, pp. 132–143, February 1998.

[14] G. P. Meyer, S. Gupta, I. Frosio, D. Reddy, and J. Kautz, "Robust model-based 3d head pose estimation," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3649–3657, December 2015.

[15] G. P. Meyer, S. Alfano, and M. N. Do, "Improving face detection with depth," *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1288–1292, March 2016.

[16] H. Nanda and K. Fujimura, "Visual tracking using depth data," *Computer Vision and Pattern Recognition Workshop*, vol. 27, no. 2, p. 37, 2004.

[17] K. Khoshelham, "Accuracy analysis of kinect depth data," *International Society for Photogrammetry and Remote Sensing Workshop Laser Scanning*, vol. 38, no. 5, p. W12, 2011.

[18] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pp. 559–568, October 2011.

[19] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," *10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, October 2011.

[20] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, third edition ed. Pearson, 2007.

[21] L. Shapiro and G. C. Stockman, *Computer Vision*, first edition ed. Pearson, 2001.

[22] A. A. Goshtasby, *2-D and 3-D Image Registration: for Medical, Remote Sensing, and Industrial Applications*. John Wiley & Sons, 2005.

[23] B. Zitova and J. Flusser, "Image registration methods: A survey," *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[24] J. Duchon, "Splines minimizing rotation-invariant semi-norms in sobolev spaces," *Constructive Theory of Functions of Several Variables*, vol. 571, pp. 85–100, 1977.

[25] J. F. Hughes, A. Van Dam, J. D. Foley, and S. K. Feiner, *Computer Graphics: Principles and Practice*. Pearson Education, 2014.

[26] L. Juan and O. Gwun, "A comparison of sift, pca-sift and surf," *International Journal of Image Processing*, vol. 3, no. 4, pp. 143–152, August 2009.

[27] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, June 2008.

[28] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the Seventh IEEE International Conference on Computer vision*, vol. 2, pp. 1150–1157, September 1999.

[29] L. C. Evans, *Partial Differential Equations*, first edition ed. Amer Mathematical Society, 1998.

[30] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *European Conference on Computer Vision*, vol. 3951, pp. 404–417, 2006.

[31] F. Crow, "Summed-area tables for texture mapping," *ACM SIGGRAPH Computer Graphics*, vol. 18, no. 3, pp. 207–212, 1984.

[32] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 4, pp. 580–585, July 1985.

[33] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, November 1998.

[34] T. Liu, A. W. Moore, A. G. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms." *Conference and Workshop on Neural Information Processing Systems*, vol. 12, pp. 825–832, 2004.

[35] V. Turau, "Fixed-radius near neighbors search," *Information Processing Letters*, vol. 39, no. 4, pp. 201–203, August 1991.

[36] P. M. Vaidya, "An o(n logn) algorithm for the all-nearest-neighbors problem," *Discrete & Computational Geometry*, vol. 4, no. 2, pp. 101–115, March 1989.

[37] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, September 1975.

[38] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, vol. 1, pp. 39–43, October 1995.

[39] A. P. Engelbrecht, *Computational Intelligence: An Introduction*. John Wiley & Sons, 2007.

[40] S. Talukder, "Mathematical modelling and applications of particle swarm optimization," Ph.D. dissertation, Blekinge Institute of Technology, 2011.

[41] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," *2007 IEEE Swarm Intelligence Symposium*, pp. 120–127, April 2007.

[42] J. F. Kennedy, J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. Morgan Kaufmann, 2001.

[43] S. A. Khan and A. P. Engelbrecht, "A fuzzy particle swarm optimization algorithm for computer communication network topology design," *Applied Intelligence*, vol. 36, no. 1, pp. 161–177, 2012.

[44] J. Peng, Y. Chen, and R. Eberhart, "Battery pack state of charge estimator design using computational intelligence approaches," *The Fifteenth Annual Battery Conference on Applications and Advances*, pp. 173–177, January 2000.

[45] T. Peram, K. Veeramachaneni, and C. K. Mohan, "Fitness-distance-ratio based particle swarm optimization," *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, pp. 174–181, April 2003.

[46] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization," *American Institute of Aeronautics and Astronautics Journal*, vol. 41, no. 8, pp. 1583–1589, August 2003.

[47] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, vol. 3, p. 1957, July 1999.

[48] F. van den Bergh and A. P. Engelbrecht, "A new locally convergent particle swarm optimizer," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 94–99, October 2002.

[49] United Nations Scientific Committee on the Effects of Atomic Radiation, "Sources, effects and risks of ionizing radiation," *United Nations Scientific Committee on the Effects of Atomic Radiation 1988 Report of the General Assembly, with Annexes*, pp. 390–390, 1988.

[50] S. B. Gokturk and C. Tomasi, "3d head tracking based on recognition and interpolation using a time-of-flight depth sensor," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. II–211 – II–217, June 2004.

[51] J. You, W. Lu, J. Li, G. Gindi, and Z. Liang, "Image matching for translation, rotation and uniform scaling by the radon transform," *Proceedings of the 1998 International Conference on Image Processing*, vol. 1, pp. 847–851, October 1998.

[52] F. Hjouj and D. W. Kammler, "Identification of reflected, scaled, translated, and rotated objects from their radon projections," *IEEE Transactions on Image Processing*, vol. 17, no. 3, pp. 301–310, 2008.

[53] M. Deshmukh and U. Bhosale, "Radon transform fourier transform approach for image matching and it's application for image registration," *International Journal of Computer Science, Software Engineering and Electrical Communication Engineering*, vol. 2, no. 1, pp. 25–30, 2011.

[54] M. Alhussein and S. I. Haider, "Improved particle swarm optimization based on velocity clamping and particle penalization," *Third IEEE International Conference on Artificial Intelligence, Modelling and Simulation*, pp. 61–64, December 2015.

[55] Q. Bai, "Analysis of particle swarm optimization algorithm," *Computer and Information Science*, vol. 3, no. 1, pp. 180–184, 2010.

[56] B. Bellekens, V. Spruyt, R. Berkvens, and M. Weyn, "A survey of rigid 3d pointcloud registration algorithms," *Fourth International Conference on Ambient Computing, Applications, Services and Technologies*, pp. 8–13, 2014.

[57] L. G. Brown, "A survey of image registration techniques," *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, 1992.

[58] N. Chumchob and K. Chen, "A robust affine image registration method," *International Journal of Numerical Analysis and Modeling*, vol. 6, no. 2, pp. 311–334, 2009.

[59] S. Du, N. Zheng, S. Ying, and J. Liu, "Affine iterative closest point algorithm for point set registration," *Pattern Recognition Letters*, vol. 31, no. 9, pp. 791–799, 2010.

[60] J. Feldmar and N. Ayache, "Locally affine registration of free-form surfaces," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*, pp. 496–501, 1994.

[61] A. W. Fitzgibbon, "Robust registration of 2d and 3d point sets," *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003.

[62] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *IEEE Transactions on Cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013.

[63] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[64] J. Ho, M.-H. Yang, A. Rangarajan, and B. Vemuri, "A new affine registration algorithm for matching 2d point sets," *IEEE Workshop on Applications of Computer Vision.*, pp. 25–25, 2007.

[65] A. Jana, *Kinect for Windows SDK Programming Guide*. Packt Publishing Ltd, 2012.

[66] Y. Jiang, T. Hu, C. Huang, and X. Wu, "An improved particle swarm optimization algorithm," *Applied Mathematics and Computation*, vol. 193, no. 1, pp. 231–239, 2007.

[67] V. Kapoor, B. M. McCook, and F. S. Torok, "An introduction to pet-ct imaging 1," *Radiographics*, vol. 24, no. 2, pp. 523–543, 2004.

[68] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[69] B. Kovács, "List mode pet reconstruction," *Sixth Hungarian Conference on Computer Graphics and Geometry, Budapest*, 2012.

[70] W. W. Moses, "Fundamental limits of spatial resolution in pet," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 648, pp. S236–S240, 2011.

[71] P. Padeleris, X. Zabulis, and A. A. Argyros, "Head pose estimation on depth data based on particle swarm optimization," *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 42–49, 2012.

[72] P. Panchal, S. Panchal, and S. Shah, "A comparison of sift and surf," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, no. 2, pp. 323–327, 2013.

[73] H. Park, G. R. Martin, and A. Bhalerao, "Local affine image matching and synthesis based on structural patterns," *IEEE Transactions on Image Processing*, vol. 19, no. 8, pp. 1968–1977, 2010.

[74] D. P. Rini, S. M. Shamsuddin, and S. S. Yuhaniz, "Particle swarm optimization: technique, system and challenges," *International Journal of Computer Applications*, vol. 14, no. 1, pp. 19–26, 2011.

[75] D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics*. McGraw-Hill Higher Education, 1989.

[76] H. M. Sahloul, H. J. D. Figueroa, S. Shirafuji, and J. Ota, "Foreground segmentation with efficient selection from icp outliers in 3d scene," *2015 IEEE International Conference on Robotics and Biomimetics*, pp. 1371–1376, 2015.

[77] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," *Proceedings of the 1999 Congress on Evolutionary Computation*, vol. 3, 1999.

[78] C. Studholme, D. L. Hill, and D. J. Hawkes, "An overlap invariant entropy measure of 3d medical image alignment," *Pattern Recognition*, vol. 32, no. 1, pp. 71–86, 1999.

[79] D. W. Townsend, "Combined positron emission tomography-computed tomography: The historical perspective," *Seminars in Ultrasound, CT and MRI*, vol. 29, no. 4, pp. 232–235, 2008.

[80] Y. Tu, C. Zeng, C. Yeh, S. Huang, T. Cheng, and M. Ouhyoung, "Real-time head pose estimation using depth map for avatar control," *Proceedings of the IPPR Conference on Computer Vision, Graphics, and Image Processing*, vol. 2, no. 4, p. 6, 2011.

[81] Y. Tu, H.-S. Lin, T.-H. Li, and M. Ouhyoung, "Depth-based real time head pose tracking using 3d template matching," *ACM SIGGRAPH Asia 2012 Technical Briefs*, p. 13, 2012.

[82] A. Watt, *3D Computer Graphics*, third edition ed. Addison-Wesley, 2000.

[83] J. Webb and J. Ashley, *Beginning Kinect Programming with the Microsoft Kinect SDK*. Apress, 2012.

[84] Z.-H. Zhan, J. Zhang, Y. Li, and H. S.-H. Chung, "Adaptive particle swarm optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 6, pp. 1362–1381, 2009.

[85] K. Zielinski, D. Peters, and R. Laur, "Stopping criteria for single-objective optimization," *Proceedings of the Third International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2005.