University of the Philippines Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

Image Navigation and Manipulation in the Operating Room using Hand Gesture Recognition

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Jaye Renzo L. Montejo

April 2014

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Image Navigation and Manipulation in the Operating Room using Hand Gesture Recognition" prepared and submitted by Jaye Renzo L. Montejo in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____

Gregorio B. Baes, Ph.D. (*candidate*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	____________	____________
2. Richard Bryann L. Chua, Ph.D. (*candidate*)	____________	____________
3. Aldrich Colin K. Co, M.Sc. (*candidate*)	____________	____________
4. Ma. Sheila A. Magboo, M.Sc.	____________	____________
5. Vincent Peter C. Magboo, M.D., M.Sc.	____________	____________
6. Geoffrey A. Solano, M.Sc.	____________	____________
7. Bernie B. Terrado, M.Sc. (*candidate*)	____________	____________

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

____

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

____

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

____

Alex C. Gonzaga, Ph.D., Dr.Eng.
Dean
College of Arts and Sciences

Abstract

In the operating room, computer-aided navigation systems are used by the surgeon to view medical images during surgery. But since sterility must be maintained at all times to prevent disease transmission and spread of infections, the surgeon cannot use the mouse, keyboard or any other controller devices in cases where there are several images that need to be viewed or if there is a need to manipulate images. In order to provide a sterile and natural human-computer interaction, this paper presents a gesture-controlled image navigation and manipulation application that uses hand gesture recognition and the computer vision capabilities of the Kinect sensor to allow the surgeon to perform navigation and manipulation on medical images.

Keywords: Human-computer interaction, hand gesture recognition, image navigation and manipulation, Kinect

Contents

List of Figures

I. Introduction

A. Background of the Study

For the past three decades, the use of computer systems in hospitals helped improve the quality of health care services and changed medical practices particularly in the area of patient data management and preoperative planning [2]. However, there are certain issues with regards to the manner in which users interact with computer systems. Controller devices like the keyboard and mouse have been found to be potential sources of disease transmission and spread of infections [3], making interaction with computers difficult in certain areas in hospitals such as operating rooms, where sterility must be observed at all times [4, 5]. Even the use of touch screens cannot prevent the spread of infections since the hands would still have physical contact with the computer. There is a need for a more natural and touchless interaction, which can be achieved through gesture recognition.

Gesture recognition is a form of natural human-computer interaction (HCI) that involves the use of gestures to control computing devices [6]. Gestures can be any bodily motion, but are commonly represented as hand movements. It enables users to interact directly with the computer without using a keyboard, mouse or any other controller device. This technology has existed since the early 1960's [6, 7] but still continues to improve until today alongside computer vision and image processing [8].

To address the problem of sterility in the operating room, several researchers developed gesture-controlled systems for navigating MRI images. Graetzel et al. [2] described such systems as a non-contact mouse for surgeon-computer interaction because it enables the surgeon to use hand gestures as an alternative to controller devices in performing standard mouse functions. Gestix by [5] is also a gesture-controlled system that can interpret hand gestures to browse medical images and is integrated to an image viewer called Gibson. The REALISM [9] on the other hand uses hand postures as commands for image navigation and manipulation.

In 2010, Microsoft introduced a motion-sensing device called the Xbox 360 Kinect which is an add-on peripheral device for the Xbox 360 video game console. It is originally intended to enhance gaming and entertainment capabilities of the Xbox 360 but since the release of the Kinect for Windows SDK, developers started building real world Kinect applications, making the Kinect enter other areas beyond gaming such as robotics [10], biometrics [11], communication [12] and health care [13, 14]. With the help of the Kinect sensor, the implementation of gesture recognition and the development of gesture-controlled systems and applications will become easier.

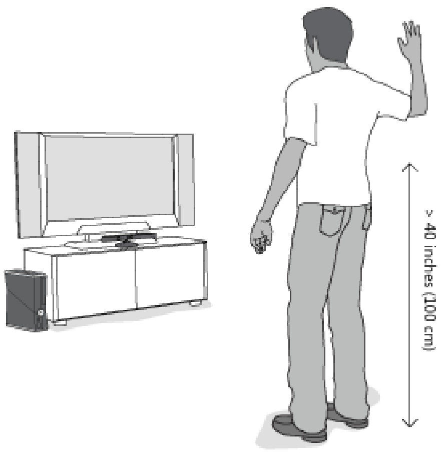


Figure 1: Kinect Play Area Setup

B. Statement of the Problem

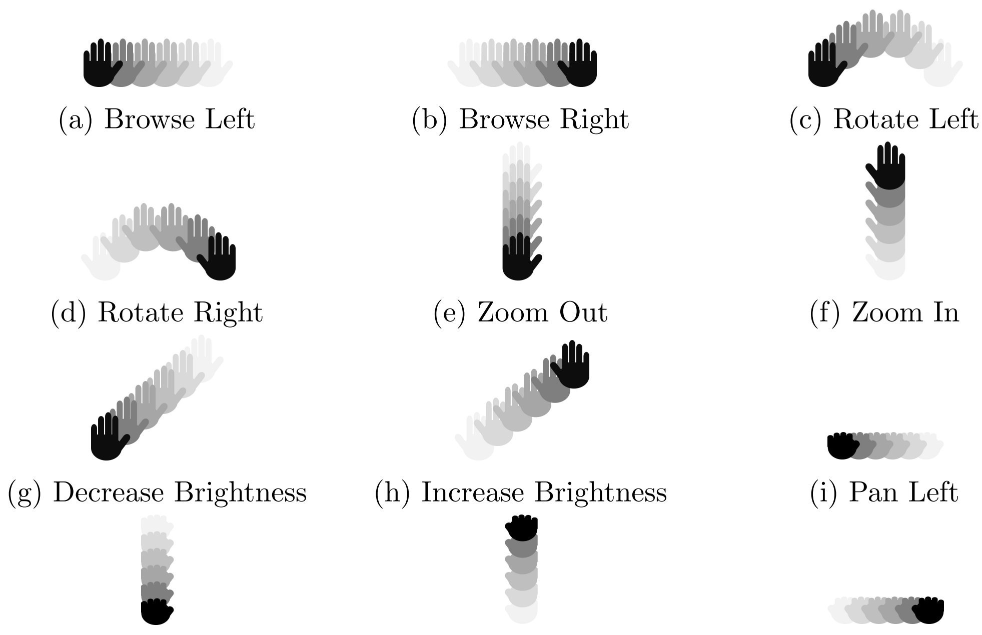
Computers are essential tools in hospitals as these help medical experts in providing fast and reliable quality health care. However, controller devices like mice and keyboards have been found to be potential sources of disease transmission and spread of infections in areas such as hospitals, where computers with multiple users are a common setting [3].

In the operating room, the surgeon needs to use a computer-aided navigation system to view medical images during surgery. These images serve as reference for the surgeon to make sure the surgery is performed correctly. However, in cases where there are several images that need to be viewed or if there is a need to manipulate images, the surgeon cannot use the mouse, keyboard or any other controller devices since he/she needs to be sterile while performing the surgery. As a result, an assistant is tasked to perform these actions on the computer while the surgeon instructs him/her on what to do. Such interactions can be cumbersome, error-prone and may slow down the whole operation since miscommunication between the surgeon and the assistant may occur. [2, 5]

We envision an operating room setting wherein the surgeon can perform the image navigation and manipulation on the computer while maintaining their sterility. In order for the surgeon to interact directly with the computer without touching any controller devices, we create a gesture-controlled image navigation and manipulation application designed for surgeon use.

C. Objectives of the Study

This research presents a gesture-controlled image navigation and manipulation application that allows the user to perform navigation and manipulation on medical images through the use of hand gestures to be able to assist during surgery.



(a) Browse Left (b) Browse Right (c) Rotate Left

(d) Rotate Right (e) Zoom Out (f) Zoom In

(g) Decrease Brightness (h) Increase Brightness (i) Pan Left

(j) Pan Down (k) Pan Up (l) Pan Right

Figure 2: The Gestural Commands

The application uses the Xbox 360 Kinect sensor to recognize gestures from the user. It has the following functionalities:

1. *Allow the user to open an image or a set of images.* The image/s are displayed on the screen and can be in .jpeg, .png, .bmp or .dcm file format.

2. *Perform skeletal tracking on the user.* It involves tracking the user's body and checking its orientation through the use of the Kinect sensor.

3. *Recognize a waving hand gesture.* A waving hand gesture tells the application that the user is ready to perform gestural commands on the images. The application assigns the control to the user who performed a waving hand gesture so that it knows whose gesture it should recognize in case two or more users are present in the viewable area.

4. *Recognize hand gestures as commands for image navigation and manipulation.* These gestural commands are shown in Figure 2. The right hand can perform browse right (2b), rotate 90° right (2d), zoom in (2f), increase brightness (2h), pan right (2l) and pan up (2k). Oppositely, the commands browse left (2a), rotate 90° left (2c), zoom out (2e), decrease brightness (2g), pan left (2i) and pan down (2j) are controlled by the left hand.

D. Significance of the Project

In terms of providing a more natural human-computer interaction, gesture recognition is an ideal approach to implement in the operating room. It will enable the surgeon to perform image navigation and manipulation during surgery without having any physical contact with controller devices, preventing disease transmission and the spread of infections. It is also more preferable than other methods of natural HCI such as speech recognition, since the voice would only add up to the high level of noise in the operating room [4]. In addition, the use of the Kinect sensor will not require the surgeon to wear any other equipment.

Assigning an assistant to handle the computer work during surgery might result to lengthy conversational exchanges between the surgeon and the assistant. This means that the surgeon will dedicate significant attention to giving orders and verifying if they are performed by the assistant correctly. In [2], it was observed that it took seven minutes and four people including the surgeon to perform a single click which was needed to configure the navigation system. With a gesture-controlled navigation system, surgical delays like this might be prevented.

Since an assistant will no longer be required, there will be a reduction of staff members in the operating room but the same amount of support will be provided. This conforms to the suggestion of [4], i.e. to maximize support at minimum staffing levels.

E. Scope and Limitations

1. The user can open an image or a set of images in .jpg, .png, .bmp or .dcm file format.

2. The user's distance from the Kinect sensor must be between 1-2 meters.

3. The user must perform a waving hand gesture to tell the application that he/she is ready to perform gestural commands on the images.

4. The user can perform gestural commands for image navigation and manipulation.

5. The application's gestural commands include browse images left/right, select image, zoom image in/out, pan image, rotate image 90° left/right and increase/decrease image brightness.

6. The application only uses one Kinect sensor.

7. The application can only recognize gestures from exactly one user.

F. Assumptions

1. Only one Kinect sensor is connected to the computer.

2. The images are opened and displayed on the screen prior to gesture recognition.

3. There are no objects present between the user and Kinect sensor that may interfere with the gesture recognition.

II. Review of Related Literature

The purpose of gesture-controlled applications vary depending on the domain in which they are used. Before Kinect was introduced in application areas beyond gaming, developers were able to implement the whole process of gesture recognition, from gesture modelling to gesture analysis and recognition, through the use of a normal video camera [6, 8]. In the following section, we will discuss various researches and developments regarding gesture recognition technology and its applications.

In the area of computer security, Gafurov [15] and Guerra-Casanova et al. [16] aimed to deviate from traditional user authentication mechanisms such as passwords, fingerprints, face and iris recognition and presented the use of hand gestures as a means of user authentication. The first reference makes use of arm swings as a way to verify the identity of the user while the second one involves user authentication by performing a one hand gesture while holding a mobile device. Shukran et al. [11] proposed the use of hand gestures for user authentication with the Kinect as the medium. They discussed various hand gesture detection algorithms and how they differ from each other in terms of accuracy, efficiency and computational intensiveness. All of these three references implemented a procedure called user enrollment. This is where the user is initially asked to perform his/her preferred hand gesture several times resulting in the generation of a gestural pattern or template which will be the basis for the succeeding verifications.

Wang et al. [17] and Malima [18] demonstrated the use of hand gestures in human-robot interaction. Each of the two references proposes a different method for handling gesture recognition but both of them focus on maintaining human-robot relationships especially in complex and uncertain environments. El-laithy et al. [10] studied about the use of Kinect in robotics applications and pointed out some limitations. First, the presence of intense sunlight may cause interference with the Kinect's Infrared Emitter, making the Kinect unable to detect objects accurately during the day. The Kinect also does not detect glass or transparent plastic because the infrared rays are refracted resulting to the Kinect returning an incorrect depth data. To handle such issues, they suggested developing new libraries that will allow calibration and modification of the Kinect sensor.

Gesture recognition is also used as a method for assisting persons with disabilities, specifically the deaf and mute in terms of communication [8]. Starner et al. [19] presented in their paper an extensible system which uses a video camera to track hands in real time and interpret American Sign Language. Soltani et al. [12] developed a Kinect gesture-based game for the deaf and mute by replacing the speech recognition feature of Microsoft's Shape Game with gestures. Applications like these will encourage the deaf and mute to engage more in social interaction and communication [8, 12].

In the area of healthcare, several researchers addressed the problem of high infection rates in hospitals due to the use of computer devices. Gesture-controlled systems for navigating MRI images were developed by [2, 5, 9, 13]. All systems serve the same purpose i.e., touchless human-computer interaction in order to maintain sterility, but each of them applied different methods.

The study of Graetzel et al. [2] is one of the early attempts to tackle the problem of sterility and it was described as a non-contact mouse for surgeon-computer interaction. They presented the idea of using hand gestures along with computer vision systems as an effective approach for surgeons to perform standard mouse functions. Gestix by Wachs et al. [5] is also a vision-based system that can interpret hand gestures to be able to browse medical images. Gestix contains the gesture interface and is integrated to an image viewer called Gibson. One of the unique features of Gestix is the use of flick gestures i.e., moving the hand rapidly out from the viewable range of the camera then back in. Another similar project named REALISM was developed by Achacon et al. [9]. The researchers of REALISM explained in detail the use of a gesture vocabulary to detect valid gestures. The REALISM gesture vocabulary consists primarily of open palm, closed fist, Y posture and L posture but the vocabulary can still be expanded to fit the user's needs making REALISM a flexible system.

Jacob et al. [13, 14] discussed about the use of visual contextual cues as a determining factor in order for the system to gauge the user's intention to interact with it. Examples of these visual cues include gaze, torso orientation, head orientation (to know if the user is facing the camera), body posture and position of the hands with respect to the body. If these visual contextual cues are not detected, the system will not be able to recognize gestural commands from the user. Once the intent has been determined, the system will then proceed to a process known as gesture spotting. Gesture spotting is the process of determining the start and end points of a gesture. Intention recognition and gesture spotting sums up the whole gesture recognition process.

Based from interviews with nine surgeons, Jacob et al. [13, 14] were able to find out the image navigation and manipulation tasks that are commonly used during surgery and the appropriate gesture for each task. These include browsing, zooming, rotating and adjusting image brightness.

Specifications that should be considered when developing a hand gesture recognition system for surgeons are presented in [5]. These specifications are as follows: (1) *Real time interaction* - the surgeon can interact with the computer even during surgery. (2) *Fatigue* - gestures should be fast and concise to reduce effort. (3) *Intuitiveness* - gestures should be cognitively related to the command it represents. (4) *Unintentionality* - unintentional gestures must be ignored by the system and intentional gestures must be recognized correctly. (5) *Robustness* - gestures in complex background and variable lighting must still be detected. (6) *Easy to learn* - gestures do not require extensive user training. (7) *Unencumbered* - since surgeons wear gloves and frequently hold instruments, additional devices attached to the hand must be avoided.

Some of the common problems and limitations encountered by these studies include: misclassification of some objects as hand, hand is in the workspace but not tracked, the need for a white background, delay between navigation and manipulation commands and low precision in a poorly illuminated environment. However, the last mentioned may be disregarded since the operating room is expected to be well-lit [2, 4].

Knowing these techniques for gesture recognition and the advantages and limitations of using Kinect as the motion sensing medium will be useful for the development of gesture-controlled applications.

III. Theoretical Framework

A. The Xbox 360 Kinect

The Xbox 360 Kinect is a motion–sensing device developed by Microsoft and is used as an add-on device for the Xbox 360 video game console. The concept of Kinect is based on Natural User Interface (NUI), i.e. interacting with the computer through gestures or voice commands instead of using a controller. [1, 20]

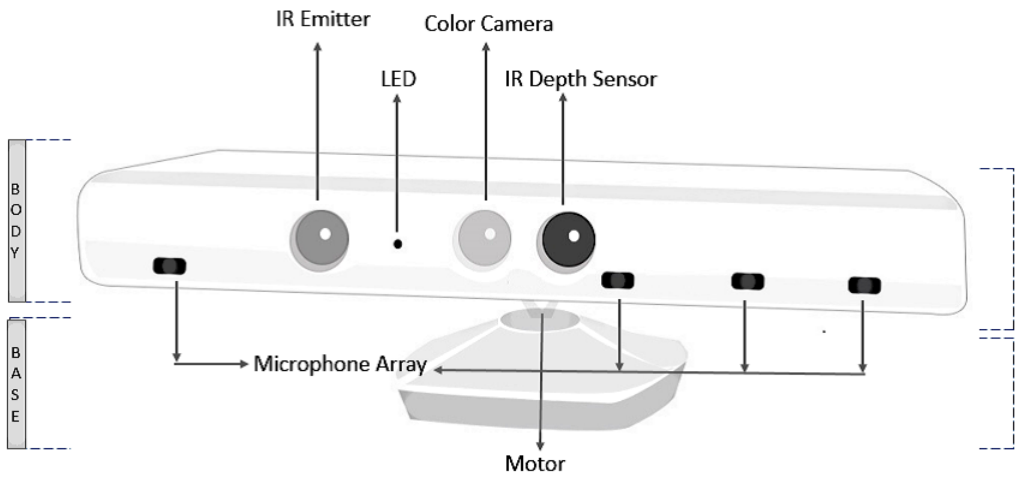


Figure 3: Components of the Kinect [1]

The Kinect has two main parts, the body and the base, as shown in Figure 3. The body contains most of the components that handle the detection of motion and voice. The base consists of a tilt motor which enables the device to be tilted in a vertical direction. Overall, the Kinect has the following key components:

1. RGB Camera or Color Camera

   The RGB Camera functions as the main component for capturing and streaming color video data. The viewable range for the RGB Camera is 43 degrees vertical by 57 degrees horizontal. [1]

2. Infrared (IR) Emitter

   The IR Emitter constantly emits infrared light in a pseudo-random dot pattern over every object located from where the Kinect is facing. The dotted light reflects off different objects and is used by the IR Depth Sensor. [1]

3. Infrared (IR) Depth Sensor

   The IR Depth Sensor locates and reads the dotted light generated by the IR Emitter. By calculating the distance between the sensor and the dotted light, it will capture the depth information of the object where the dotted light was located. [1]

4. Microphone Array

   The Microphone Array consists of four microphones that capture the sound and its location. The microphones handle the sound and voice recognition. [1]

5. LED

   The LED is used to indicate the status of the Kinect sensor. The green color tells the user that the computer has detected the Kinect and that its drivers have loaded properly. [1]

6. Tilt Motor

   The body and the base part of the Kinect is connected by the Tilt Motor. It is used to adjust the sensor's angle so that the Kinect is facing the user's desired viewable area. Through the Tilt Motor, the body of the Kinect can be shifted upwards or downwards by 27 degrees. [1]

Aside from these components, the Kinect also requires a power adapter for external supply and a USB adapter in order to be connected to a computer.

B. The Kinect for Windows SDK

The Kinect for Windows Software Development Kit was first released as a beta version by Microsoft on June 2011. It consists of certain development tools that can be used for developing Kinect applications in C#, C++ or VB.NET. Upon installation of the Kinect for Windows SDK, an additional toolkit called Kinect for Windows Developer Toolkit can also be installed. The toolkit includes Kinect programming samples, documentations, tutorials and other development extensions.

Although the Kinect for Windows SDK will aid the developer in making Kinect applications, it does not completely provide any built-in Application Programming Interfaces (API's) exclusive for gesture recognition. This means that it will be up for the developer to define and formulate his/her own approaches for recognizing gestures.

C. Skeletal Tracking

Skeletal Tracking is one of the features included in the Kinect for Windows SDK. It basically allows the Kinect to track the human body and follow its motion and actions. To be able to create gesture-controlled applications using Kinect, it is necessary to have at least a basic understanding on how skeletal tracking works.

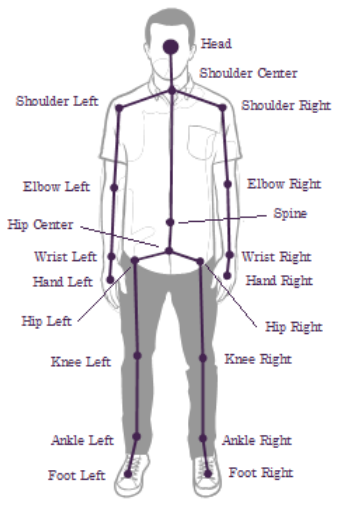


Figure 4: The twenty joint points [1]

If a human body becomes visible within the viewable range of the Kinect, the IR Emitter and IR Depth Sensor work in pairs to be able to detect the human body. On this process however, the Kinect only recognizes the human body as simply an object and is only considered as a raw depth data, just like any other non-human object the Kinect detects. To be able to recognize the object as a human body, the Kinect will compare each pixel of the raw depth data with that of the machine-learned data. The machine-learned data consists of several character models with different human body characteristics. Once the raw depth data matches with the machine-learned data, the Kinect will now recognize the object as a human body. [1]

After the matching process, the Kinect will now identify the parts of the human body by using decision tree learning. Once the identification is done, it will now position the joint points all throughout the body. These joint points will now serve as the indicators of the Kinect sensor in order to track the movement of the whole body. [1]

The Kinect for Windows SDK supports detection of up to six players, only two of which can be tracked in detail. This means that it can track the twenty joint points from each of the two users but can only detect the positions of the remaining four. Each joint point is identified by a unique name as shown in Figure 4. The SDK provides a set of APIs for easy access to these joints. By using these joint points, we can now formulate our logic and create algorithms for recognizing gestures.

D. Approaches to Gesture Recognition

Approaches to gesture recognition vary depending on the kind of gestures chosen and how these gestures are used in the application. For this project, two approaches will be used.

D.1 Algorithmic Gesture Recognition

In gesture recognition, a performed gesture is checked by matching it on some pre-defined set of conditions known as the result set. Conditions in this result set may involve checking the joint point's location or position or comparing the distance or deviation between two or more joints. This approach involves playing around with the joint points, which are represented by the x, y and z coordinates and applying simple calculations to determine a valid gesture. [1]

Suppose we want to determine if the user's hands are close to each other. To do this we need to calculate the distance between the left and the right hand joint. We represent the left and right hand joints as two points P_1 and P_2 in a three dimensional plane with coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) respectively. Using the distance formula (Eq. 1), we can find the distance d of P_1 and P_2.

$$d = \|P_2 - P_1\| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{1}$$

Once the distance is calculated, we can compare it to some small value, say 0.1 to check whether both hands are already close to each other.

Some gestures however, cannot be recognized by simply calculating the distance between points. For example, to determine if both hands are raised we need to check if the y values of the left and the right hand joints are greater than that of the head joint. We can set a positive threshold value t and check if conditions $y_{\text{lefthand}} > y_{\text{head}} + t$ and $y_{\text{righthand}} > y_{\text{head}} + t$ are satisfied.

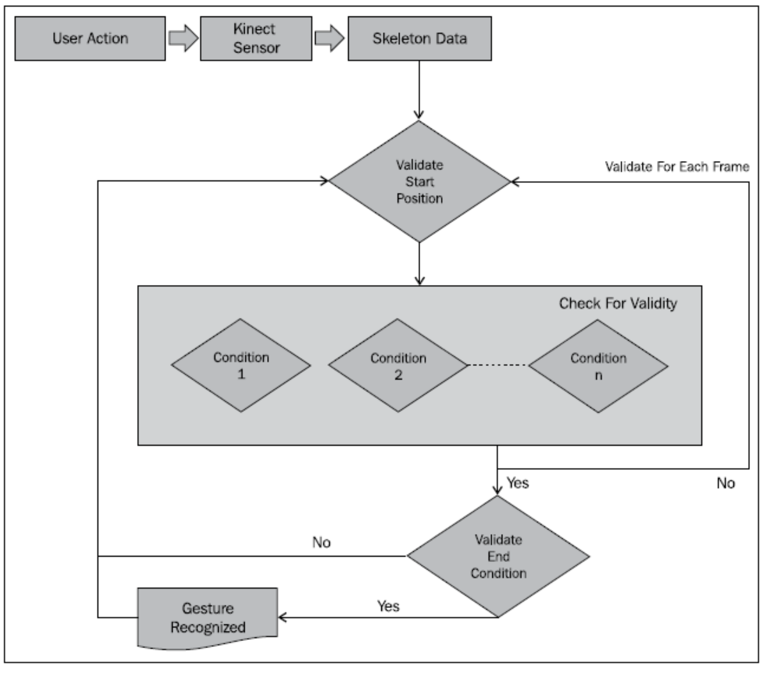


Figure 5: Algorithmic Gesture Recognition [1]

Gestures are considered to be measured by an algorithmic approach when there is a need to use multiple joints and the gestures need to be validated against multiple conditions and measured at multiple states. In algorithmic approach, we do not just check if the conditions are satisfied, we also track the user's movement for each and every frame to check if the gesture is performed correctly or not. [1]

To recognize a gesture algorithmically, we first validate its initial position. This will be the entry point for any gesture and has to be validated before other positions. Once the initial position is validated, we need to check every frame to determine whether the user is still on the process of performing the gesture. If the conditions for the gesture are all satisfied, we validate its final position which concludes the gesture recognition process (Figure 5). [1]

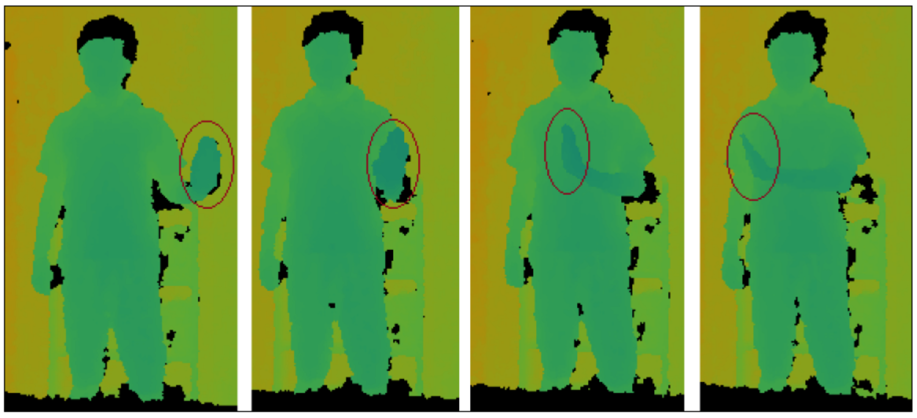


Figure 6: Swipe Left Gesture [1]

For example, to validate a swipe to left gesture (Figure 6) we need to check the following conditions:

1. The left hand joint is below the left elbow joint and the right hand joint is below the right shoulder joint and above the right elbow joint. This is the initial state.

2. The right hand joint is moving from right to left while maintaining the positions of other joints. To validate this, we need to check if the distance between the right hand and the left elbow joint is decreasing in every frame.

3. After a specific number of frames, we check if the distance between the right hand and the left shoulder joint is less than that of the initial position. This will be the final state.

If for a single frame a particular condition is not satisfied, we stop the whole process and wait for the user to perform the gesture again from its initial position. In general, the validation of a gesture is done on each and every frame so it is necessary to set a maximum frame number or timestamp for each gesture.

D.2 Template-based Gesture Recognition

In template-based gesture recognition, the user's movements are taken as input parameters and the gesture recognition engine tries to find a match from a predefined set of gestures known as patterns or templates [1]. The main concept in template matching involves having a database of templates that will be used to compare with the input gesture obtained from the user. Then a pattern matching algorithm determines if one of the templates matches the input gesture and if so, then the gesture is detected as valid [21].

A template-based recognition system involves three phases [1]:

1. Template Creation

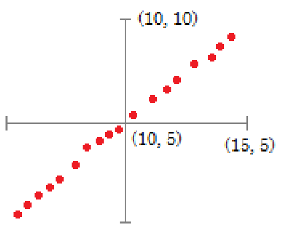
   In this phase, gesture templates are recorded and stored in memory. A gesture template may consist of several data including a sequence of points that define the movement of the gesture, the minimum and maximum length and duration values, the name of the gesture and so on.

2. Gesture Tracking

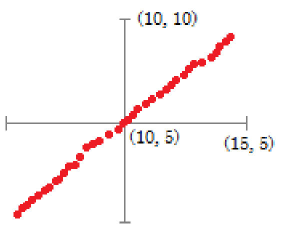
   This is where the input gesture is taken from the user.

3. Template Matching

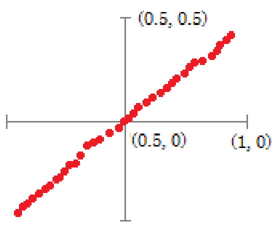
   The input is then normalized and compared to each of the gesture templates via a pattern matching algorithm. If there is a match, then a valid gesture is detected.



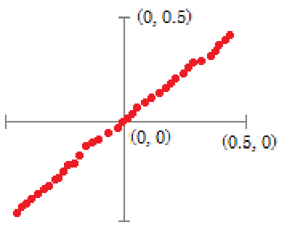
(a) Obtain input gesture from user



(b) Modify so it contains a fixed number of points



(c) Rescale to a 1x1 reference graduation



(d) Center to origin

Figure 7: Data Normalization Process

Before attempting to find a match from the set of gesture templates, we first need to normalize the input data. A gesture basically consists of a sequence of points. However, the coordinates of these points are relative to the position of the sensor i.e., every point is expressed in a coordinate system in which the sensor is at $(0, 0)$. Also, every time a user performs a gesture, the number of points that can be obtained varies. And so we need to bring the points to the same reference the gesture templates use and make the number of points the same. Having the same reference, same number of points and same bounding box makes it easier to compare the input to each of the templates [21]. The normalization process is shown in Figure 7.

After normalizing the input, we can now compare it to each of the gesture templates and see if we can get a match. To do this, we need to find the nearest neighbor of each point P_i of the input gesture by calculating its distance (Eq. 1) from point $P_{j,i}$ of template j. An example is shown in Figure 8.

Furthermore, we need to convert the measure from distance to probability so that it is easier to decide which among the gesture templates is closest to the input gesture. If the distance between two points is 0, then the probability is high.

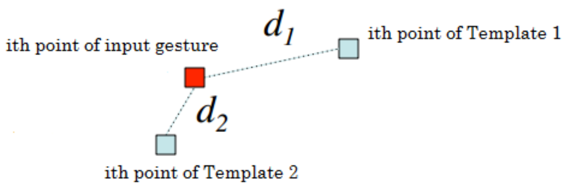


Figure 8: Comparing Distances

If the distance approaches ∞, then the probability is 0. This can be computed via the Gaussian window function [22, 23] which uses exponentiation. The total probability that the input gesture is approximately equal to gesture template j is given by Eq. 2.

$$Q_j = \frac{1}{n_j} \sum_{i=1}^{n_j} \exp\left(-\frac{\|P_{j,i} - P_i\|^2}{2}\right) \tag{2}$$

The value n_j is the number of points on template j. $P_{j,i}$ is the *ith* point of template j and P_i is the *ith* point of the input gesture. Since we modified the input gesture to have a fixed number of points during the normalization process, we are sure that the number of points of the input gesture is equal to n_j. We apply this to every gesture template j in the set and determine which among the templates has the highest total probability. The gesture template with the highest total probability Q is more likely to be the gesture performed by the user.

IV. Design and Implementation

A. Use Case Diagram

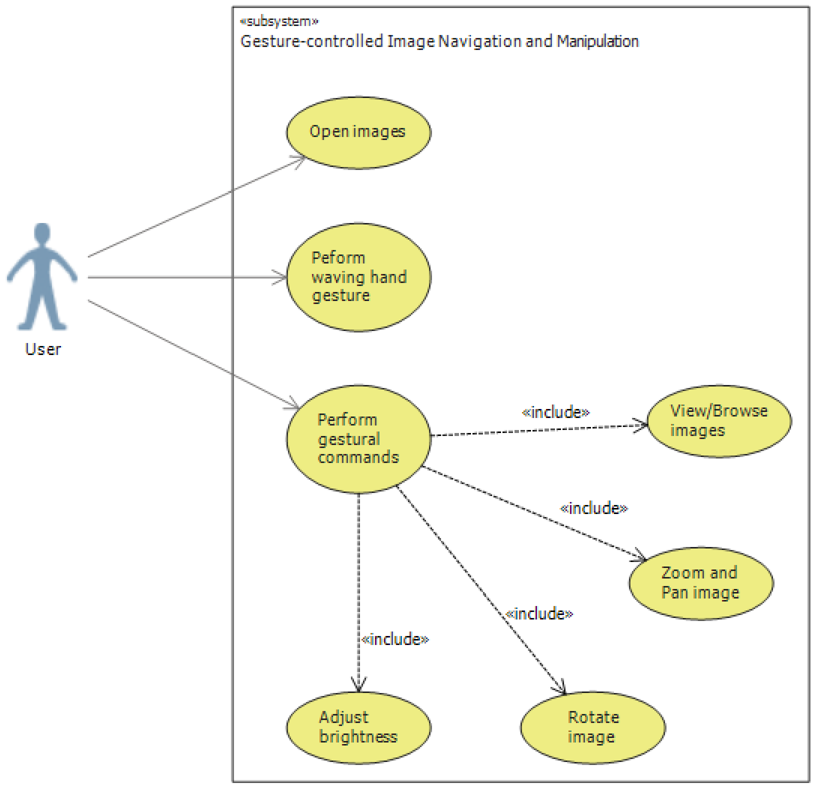


Figure 9: Use Case Diagram

The Use Case Diagram is shown in Figure 9. There is only one actor or user for this application, which represents the surgeon. The user has associations to three use cases namely *Open images*, *Perform waving hand gesture* and *Perform gestural commands*. These use cases represent the three main functionalities the user can carry out on the application. It is necessary to point out that these functionalities should be performed in sequence, starting with the opening of images. When the images are loaded and displayed, the user can now perform a waving hand gesture to tell the application that he/she is ready to perform gestural commands. These gestural commands are represented as the four use cases, *View/Browse images*, *Zoom and Pan image*, *Rotate image* and *Adjust brightness* connected to the use case *Perform gestural commands* through include relationships.

B. Class Diagram

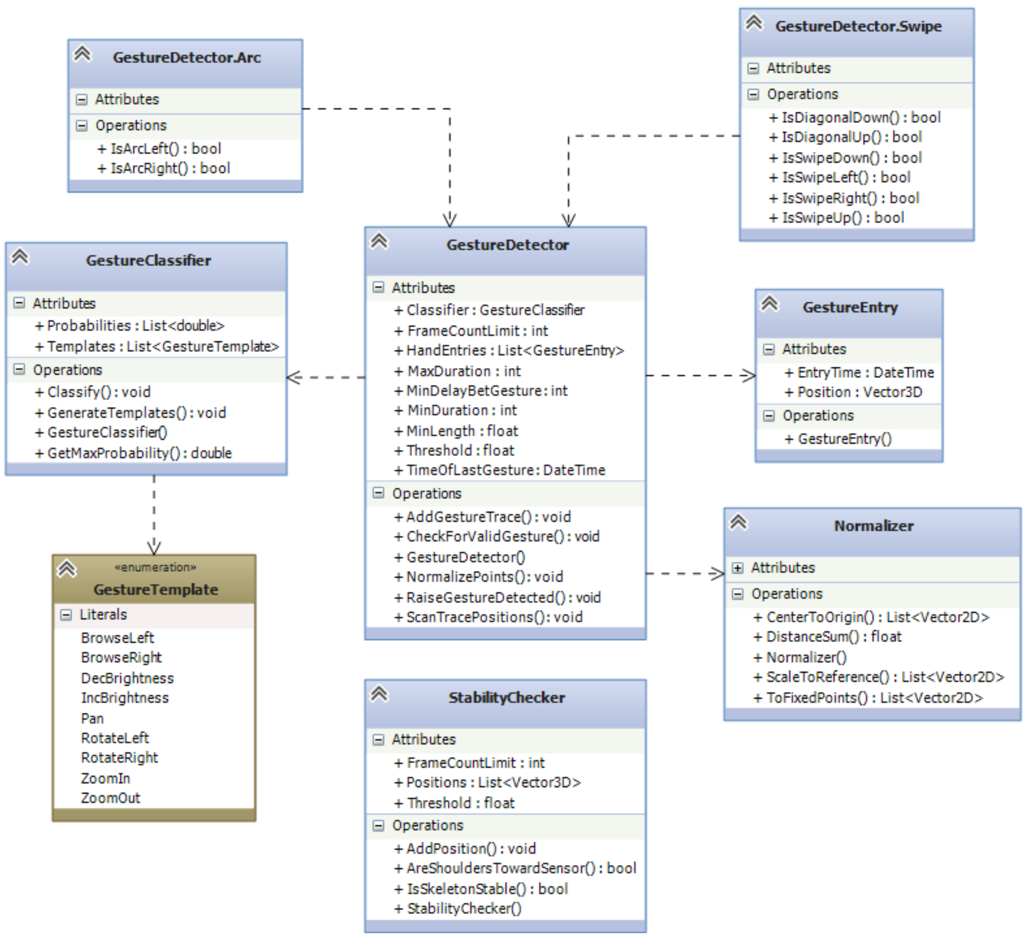
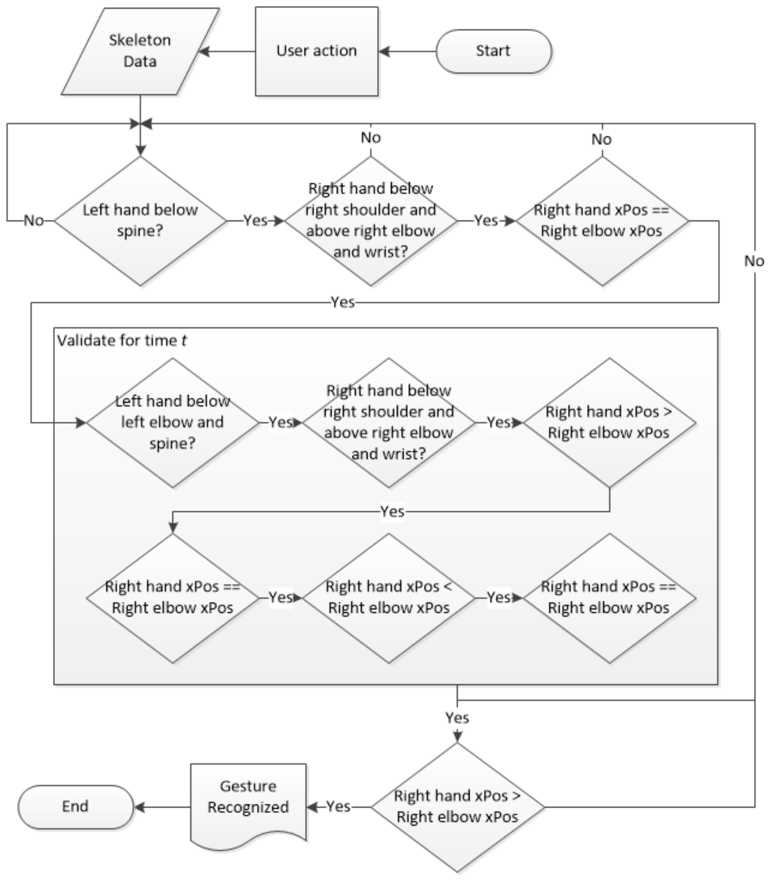


Figure 10: `GestureRecognizer` Class Diagram

The class diagram for `GestureRecognizer` is shown in Figure 10. The `GestureTemplate` enumeration contains the list of the types of gestures. The `GestureClassifier` class generates the templates of the gestures listed on `GestureTemplate`. It also has the methods for classifying and computing the probabilities for each gesture. The `GestureDetector` class connects the `GestureRecognizer` to the main application. This class contains the methods for getting the input gesture and determining what gesture has been performed. It notifies the main application what type of gesture is recognized through the `RaiseGestureDetected()` method. The `GestureDetector.Swipe` and `GestureDetector.Arc` classes are extensions of `GestureDetector`. For checking the user's stability, the `StabilityChecker` class is used.

The `Normalizer` class is a static class that contains the methods necessary for data normalization. The methods `ToFixedPoints()`, `ScaleToReference()` and `CenterToOrigin` are the main processes for data normalization and these methods are applied to the input gesture before classification. It is used by the `GestureDetector` class.

C. Flow Chart



Start
User action
Skeleton Data
Left hand below spine?
No
Yes
Right hand below right shoulder and above right elbow and wrist?
No
Yes
Right hand xPos == Right elbow xPos
No
Yes
No
Validate for time *t*
Left hand below left elbow and spine?
Yes
Right hand below right shoulder and above right elbow and wrist?
Yes
Right hand xPos > Right elbow xPos
Yes
Right hand xPos == Right elbow xPos
Yes
Right hand xPos < Right elbow xPos
Yes
Right hand xPos == Right elbow xPos
Yes
Right hand xPos > Right elbow xPos
Yes
Gesture Recognized
End

Figure 11: Wave gesture algorithm

We can divide the process of recognizing a wave gesture by three segments. For example, on determining whether a wave gesture has been performed by the right hand, the first segment should have the x position of the right hand joint be greater than the x position of the right elbow joint. On the second segment, the x position of the right hand joint should now be less than that of the right elbow joint. Lastly, on the third segment, the right hand x position should be again greater than that of the right elbow joint. When the conditions of these three segments are satisfied, a wave gesture is detected. It should be noted that this movement is checked on consecutive frames. The process is shown in Figure 11.

In checking the user's stability, we should determine the position of the user for n consecutive frames. Then we compute for the average position by summing all positions from the $1st$ to the $(n-1)th$ frame. By getting the distance between the average position and the position on the nth frame and checking whether it exceeds a given threshold t, we can determine whether a user is stable or not. The point here is that if the distance between the average position and the position on the nth frame is too large, the user is probably moving too much and is therefore not yet ready to use the application.

We also need to check if the user is facing the sensor. To do this, we check if both z positions of the left and right shoulder joints are approximately equal. The whole process of checking the user's stability is shown in Figure 12.

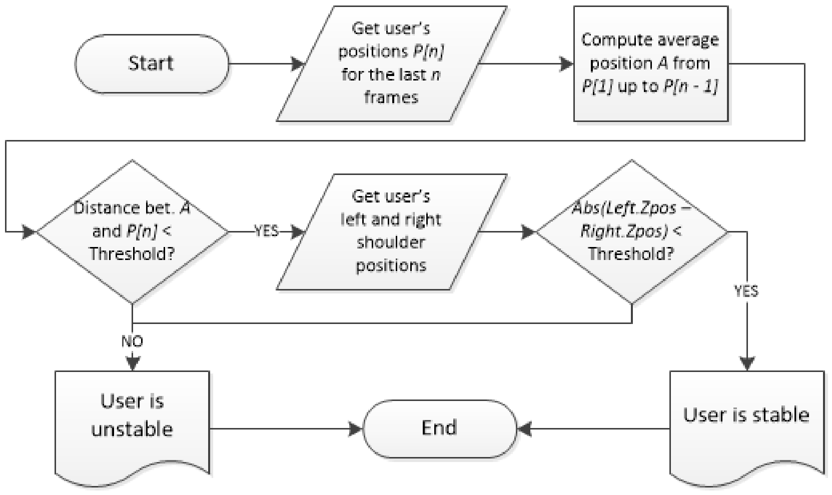

Start
Get user's positions P[n] for the last n frames
Compute average position A from P[1] up to P[n - 1]
Distance bet. A and P[n] < Threshold?
YES
Get user's left and right shoulder positions
Abs(Left.Zpos – Right.Zpos) < Threshold?
YES
NO
User is unstable
End
User is stable

Figure 12: Stability checker algorithm

In every gesture, we first check the starting conditions, i.e. the left/right hand is below the left/right elbow and spine and the right/left hand is below the right/left shoulder and above the right/left elbow and wrist. We also need to check if the z positions of the gesture points are not changing. There are also additional starting conditions that are unique to every gesture. For example, for the browse right and rotate right gestures, we need to check whether the x values are increasing. The zoom in gesture on the other hand, requires increasing y values.

After validating the starting conditions, we then normalize the sequence of points of the input gesture to prepare them for template matching. We can classify gestures into four templates, namely horizontal swipe, vertical swipe, diagonal swipe and arc. Each template has its corresponding probability whose value depends on the input gesture. The template with the highest probability will then be chosen as the recognized gesture. After this process, an event is raised, telling the main application what image command needs to be executed. Figure 13 shows the template matching process.

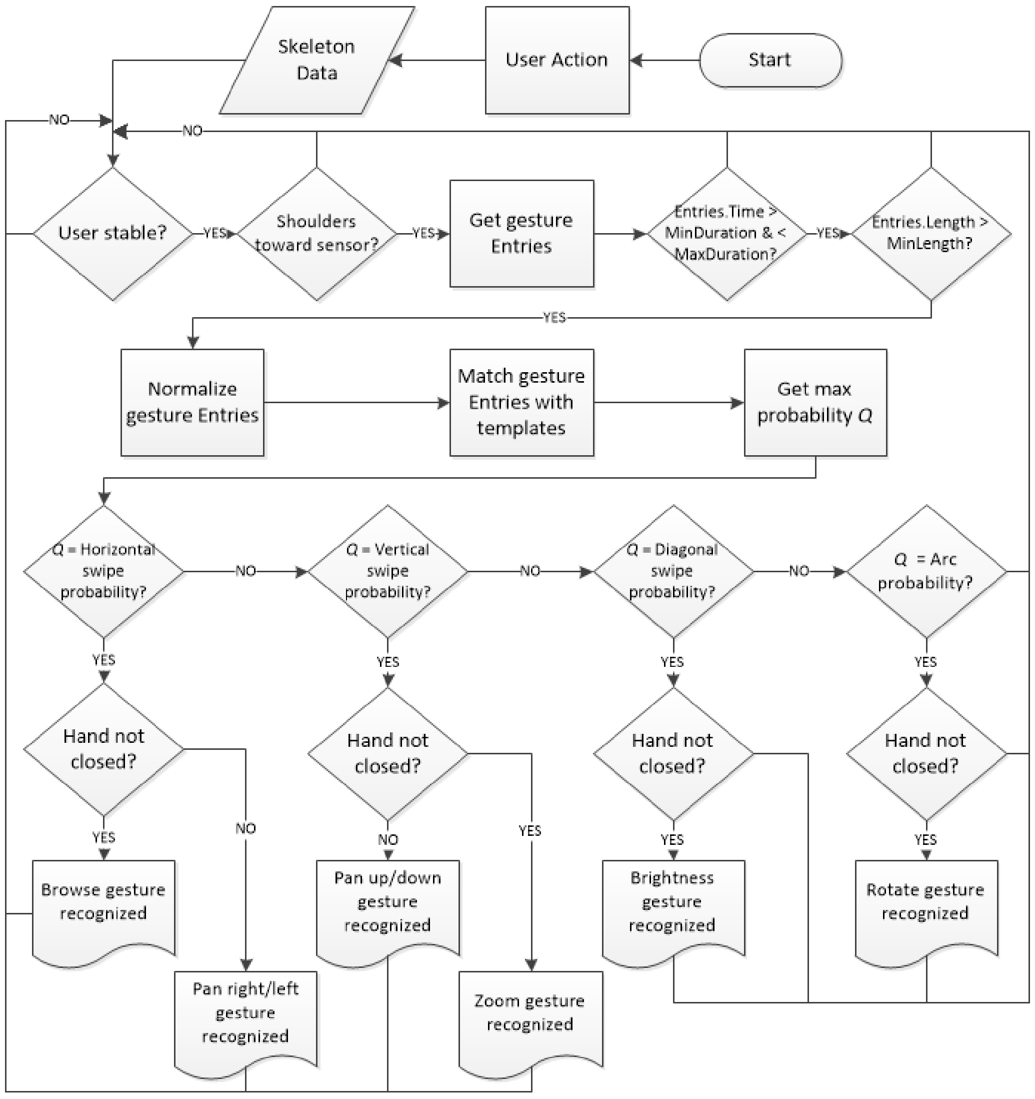

Start
User Action
Skeleton Data
NO
NO
User stable?
YES
Shoulders toward sensor?
YES
Get gesture Entries
Entries.Time > MinDuration & < MaxDuration?
YES
Entries.Length > MinLength?
YES
Normalize gesture Entries
Match gesture Entries with templates
Get max probability Q
Q = Horizontal swipe probability?
NO
Q = Vertical swipe probability?
NO
Q = Diagonal swipe probability?
NO
Q = Arc probability?
YES
Hand not closed?
YES
NO
Browse gesture recognized
Pan right/left gesture recognized
Pan up/down gesture recognized
Zoom gesture recognized
Brightness gesture recognized
Rotate gesture recognized

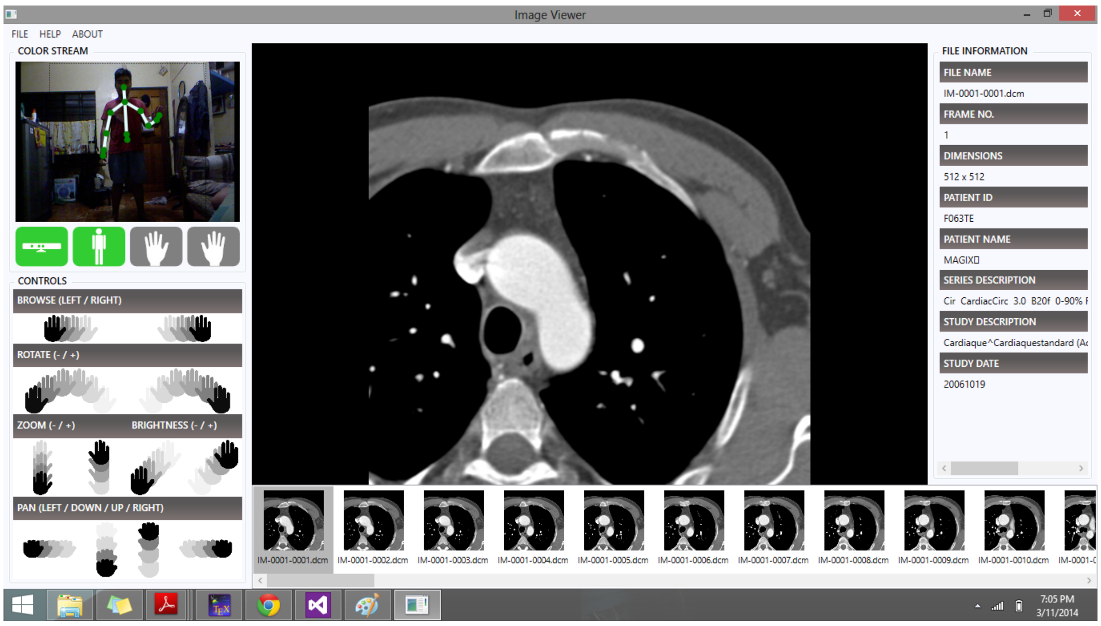
Figure 13: Template matching algorithm

D. Technical Architecture

The user must meet the following system requirements to be able to run this application.

1. Supported Operating Systems
    (a) Windows 7
    (b) Windows 8
    (c) Windows Embedded Standard 7
    (d) Windows Embedded Standard 8
2. Hardware Requirements
    (a) Microsoft or Xbox 360 Kinect sensor
    (b) 32-bit (x86) or 64-bit (x64) processors
    (c) Dual-core, 2.66 GHz or faster processor
    (d) USB 2.0 bus dedicated to the Kinect sensor
    (e) Minimum of 2 GB of RAM
    (f) Graphics card that supports DirectX 9.0c
3. Software Requirements
    (a) .NET Framework 4 or higher
    (b) Kinect Runtime Setup

V. Results



FILE INFORMATION

FILE NAME
IM-0001-0001.dcm

FRAME NO.
1

DIMENSIONS
512 x 512

PATIENT ID
F063TE

PATIENT NAME
MAGIX□

SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90% F

STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (Ad

STUDY DATE
20061019

Figure 14: Main User Interface

The main user interface of the application is shown in Figure 14. It is divided into five panels:

1. Main Image View

   This is where the currently selected image or frame is displayed. It is located in the center of the user interface.

2. Color Stream Panel

   The color stream panel contains the video stream captured by the Kinect sensor along with five visual indicators. The first one is the skeleton, which will appear on the video stream when a user is visible on the viewable area. The second one is the Kinect status indicator, represented by the Kinect icon. A green color means the sensor is online and ready; yellow means the sensor is initializing and a red one means the sensor is disconnected. The third one is the user status indicator. It tells the user whether he/she is stable (green) or not (red). The last two indicators show whether the hands are closed or open.

3. Controls Panel

   The gestures that can be performed by the user along with their corresponding functions are shown here.

4. Image List View

   This is where the thumbnails of the images or frames are displayed in case the user loads several images. The thumbnail of the image displayed in the Main Image View are highlighted with a gray color.

5. File Information Panel

   This panel displays some information regarding the image. These include the image file name, frame number, dimensions, patient ID, patient name, series description, study description and date.

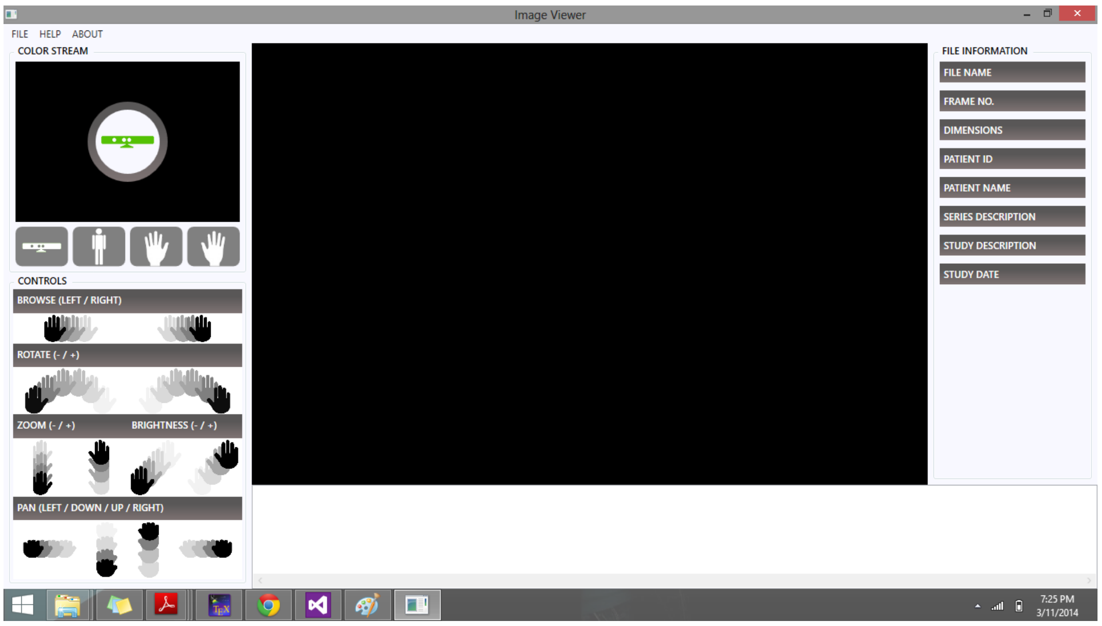
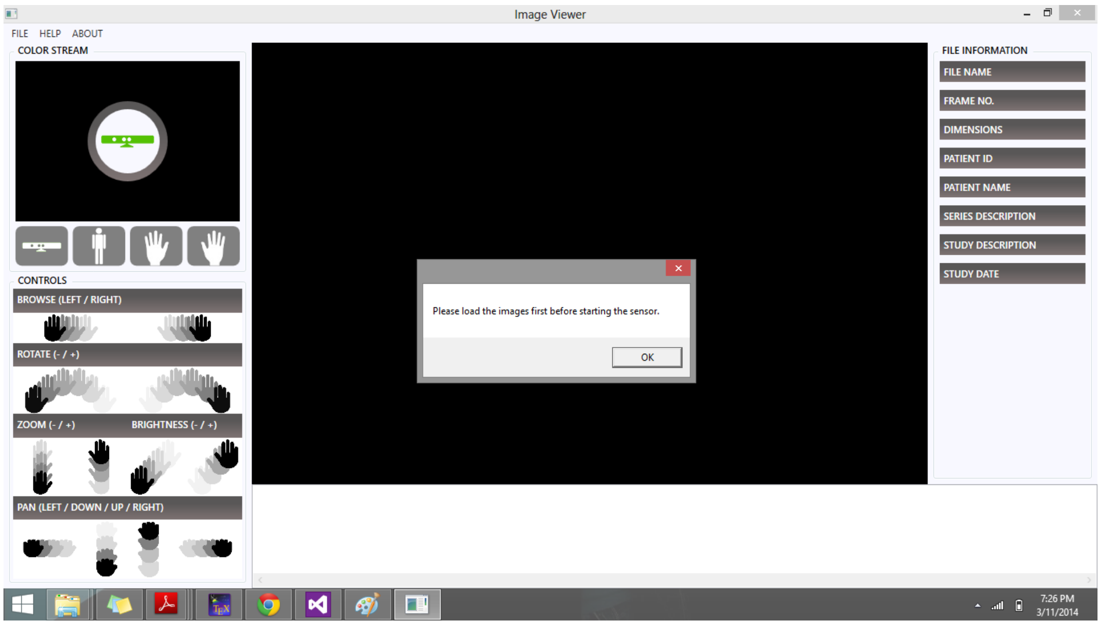


Figure 15: User interface at start up

The user interface at start up is shown in Figure 15. By clicking the circle button on the Color Stream panel, the Kinect sensor can be initialized and the video will start streaming. However, the application requires that the user must first load the images before starting the sensor. Otherwise, a message will pop-up as shown in Figure 16.



Please load the images first before starting the sensor.

Figure 16: No image loaded

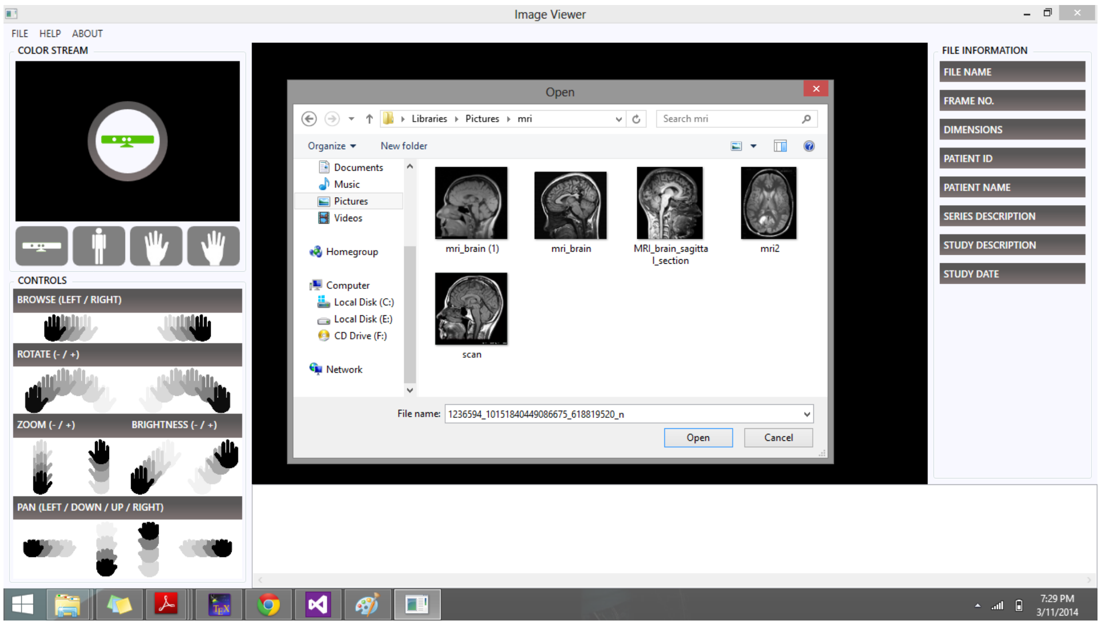
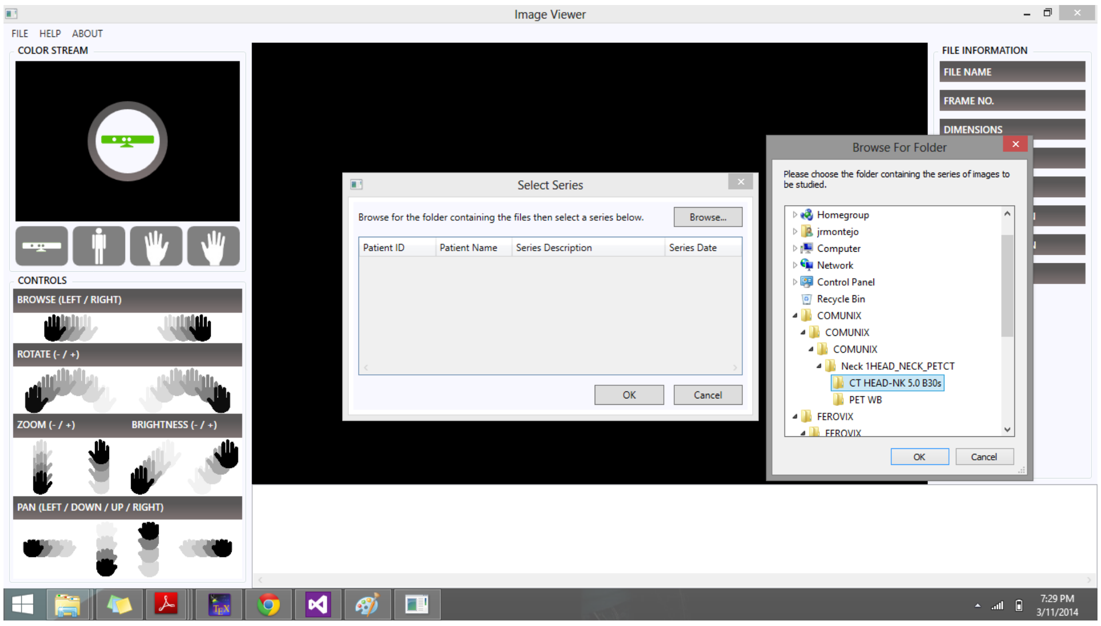


Figure 17: Open image dialog box

There are two ways to open an image or a set of images. The first one is to select one or more images via the default file chooser dialog box (Figure 17). The file types of images that can be opened in this manner are .jpg, .png and .bmp.

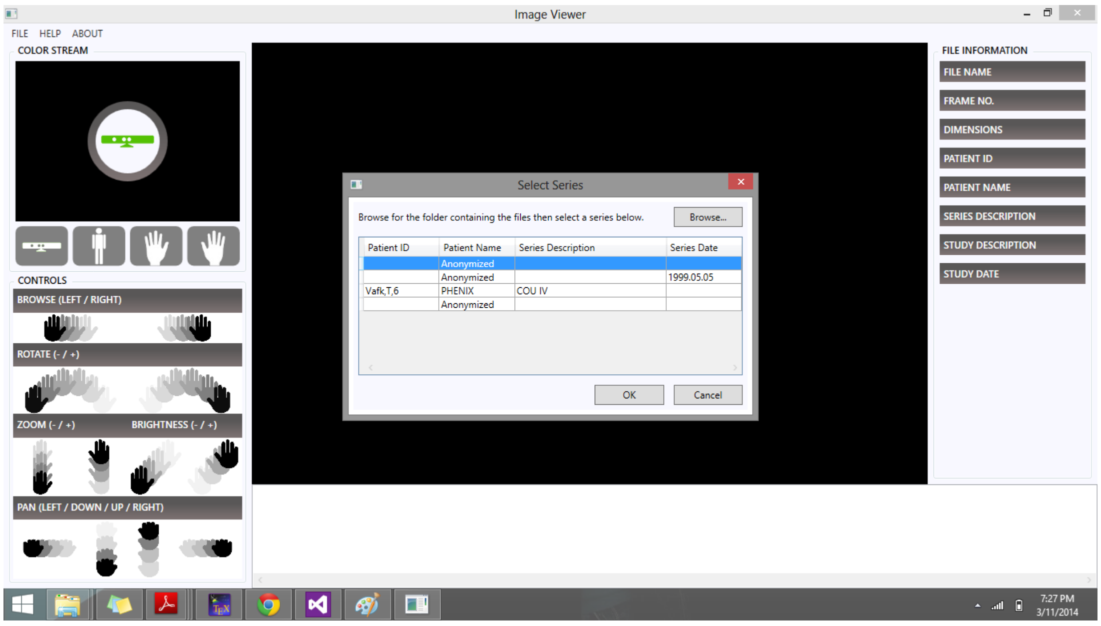


Browse for the folder containing the files then select a series below.

Patient ID	Patient Name	Series Description	Series Date

Please choose the folder containing the series of images to be studied.

Figure 18: Open folder dialog box

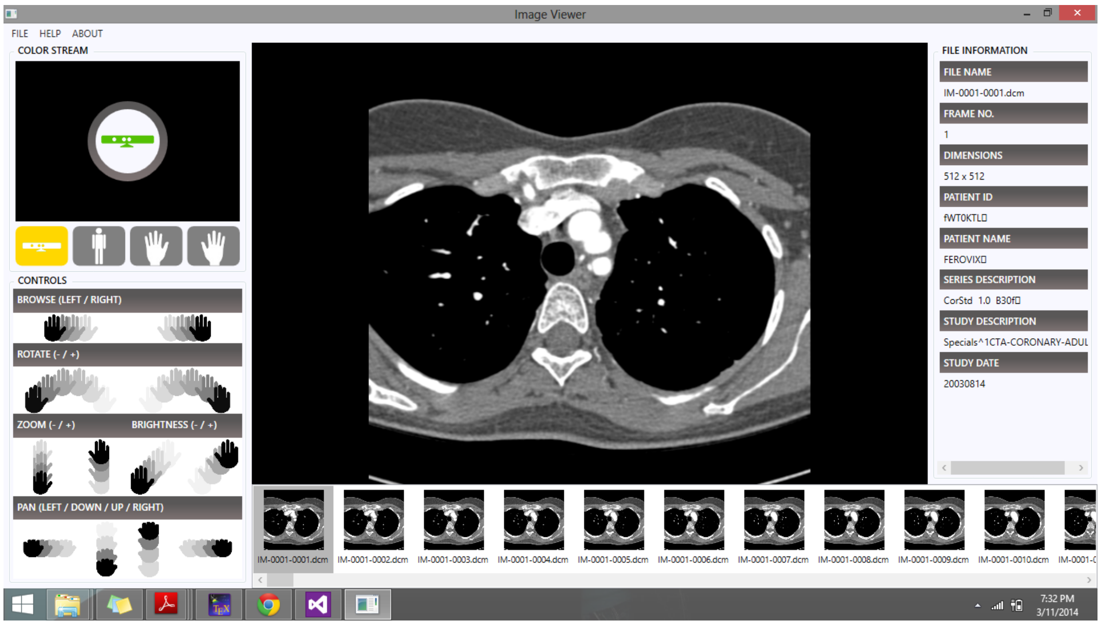


Browse for the folder containing the files then select a series below.

Patient ID	Patient Name	Series Description	Series Date
	Anonymized		
	Anonymized		1999.05.05
Vafk,T,6	PHENIX	COU IV	
	Anonymized		

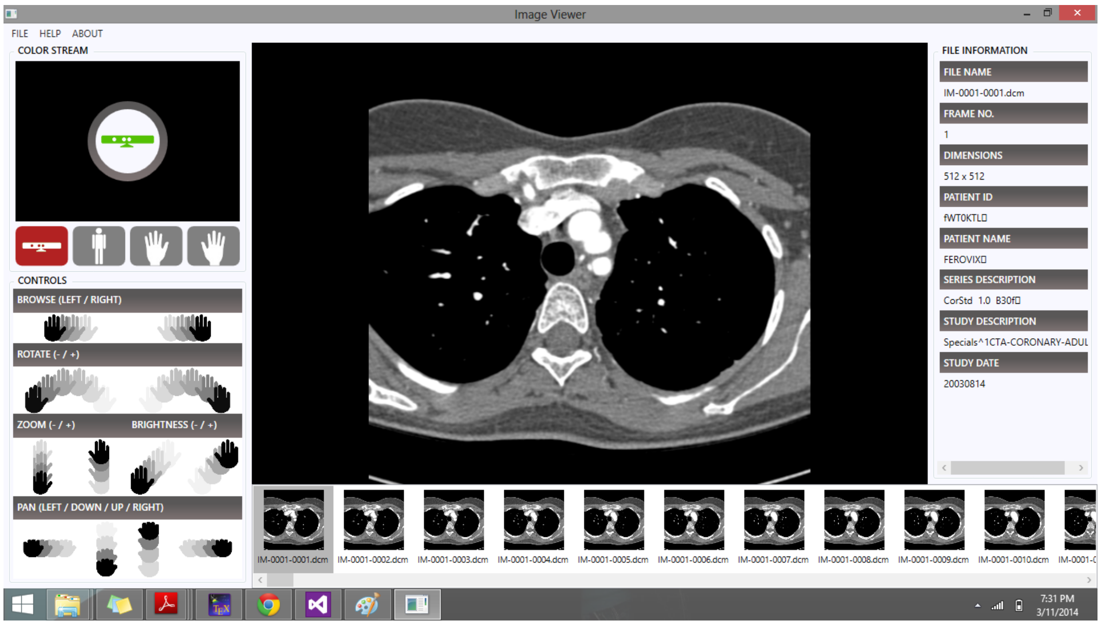
Figure 19: Select series dialog box

DICOM images (.dcm) however, are special types of images that needs to be loaded by series. One file may contain several image frames thus, DICOM files are normally grouped depending on their SeriesUID. To load DICOM images, the user must choose a folder containing the series of images (Figure 18). The series chooser dialog box will display all information on each series and the user must select only one from them by clicking on a row (Figure 19).



FILE NAME	IM-0001-0001.dcm
FRAME NO.	1
DIMENSIONS	512 x 512
PATIENT ID	fWTOKTL□
PATIENT NAME	FEROVIX□
SERIES DESCRIPTION	CorStd 1.0 B30f□
STUDY DESCRIPTION	Specials^1CTA-CORONARY-ADUL
STUDY DATE	20030814

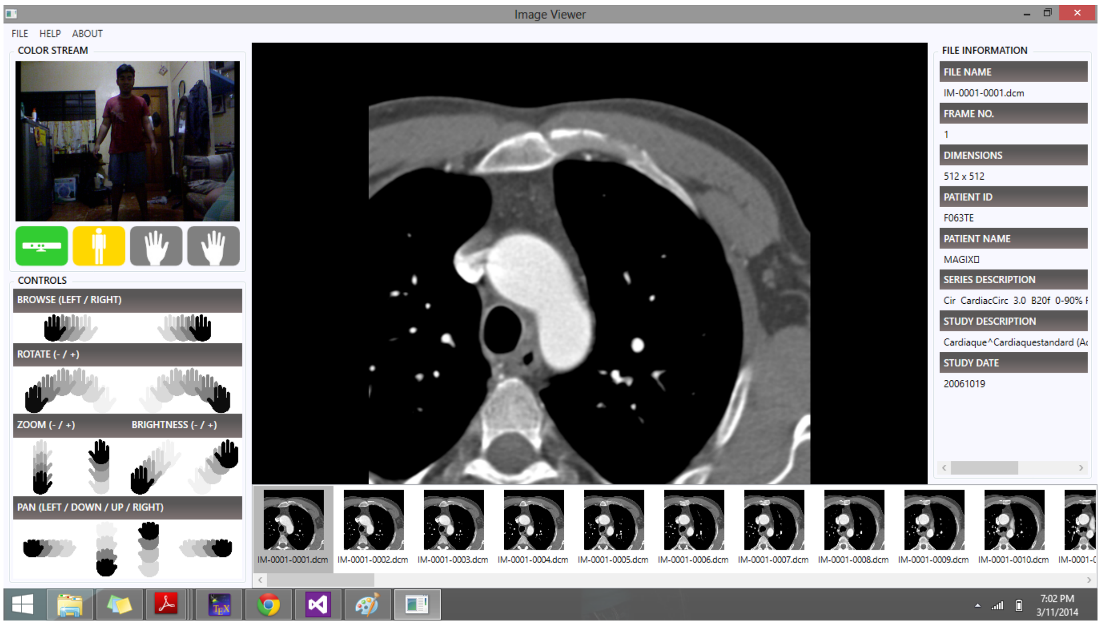
Figure 20: Kinect is initializing



FILE NAME	IM-0001-0001.dcm
FRAME NO.	1
DIMENSIONS	512 x 512
PATIENT ID	fWTOKTL□
PATIENT NAME	FEROVIX□
SERIES DESCRIPTION	CorStd 1.0 B30f□
STUDY DESCRIPTION	Specials^1CTA-CORONARY-ADUL
STUDY DATE	20030814

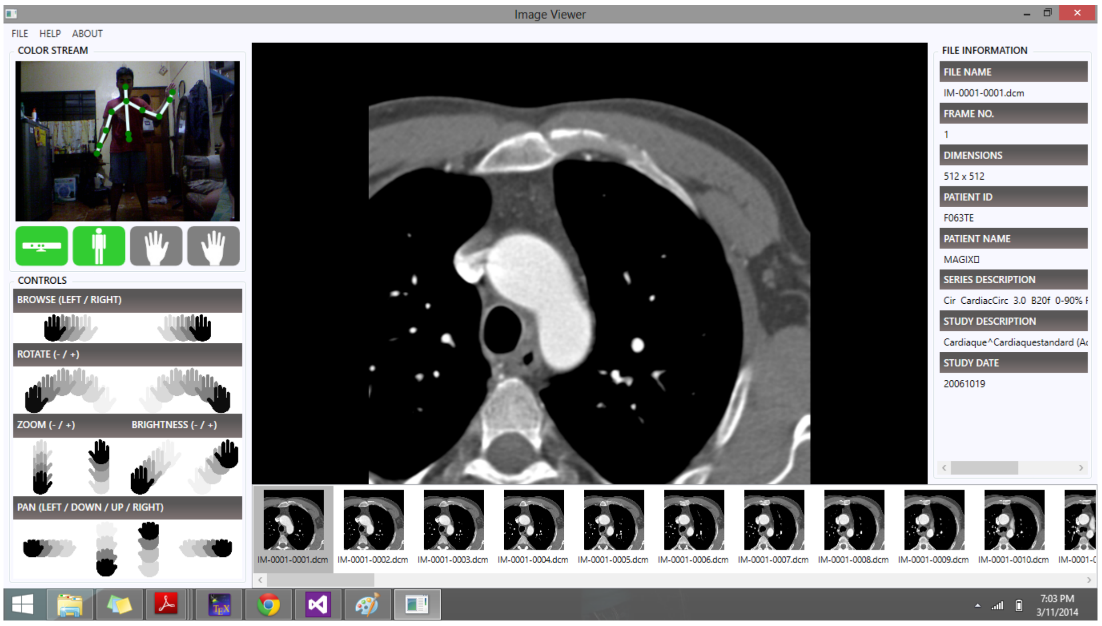
Figure 21: No Kinect detected

In the two figures above, the set of images has been loaded on the main interface. When the user tries to click the Kinect button, there are three events that can happen. If the Kinect sensor is connected to the computer but is still on the process of loading resources, the Kinect status indicator will turn yellow (Figure 20). If the sensor is disconnected, the indicator will turn red (Figure 21). When the sensor is ready, the indicator will turn green and the video will start streaming.



FILE NAME
IM-0001-0001.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIXO
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

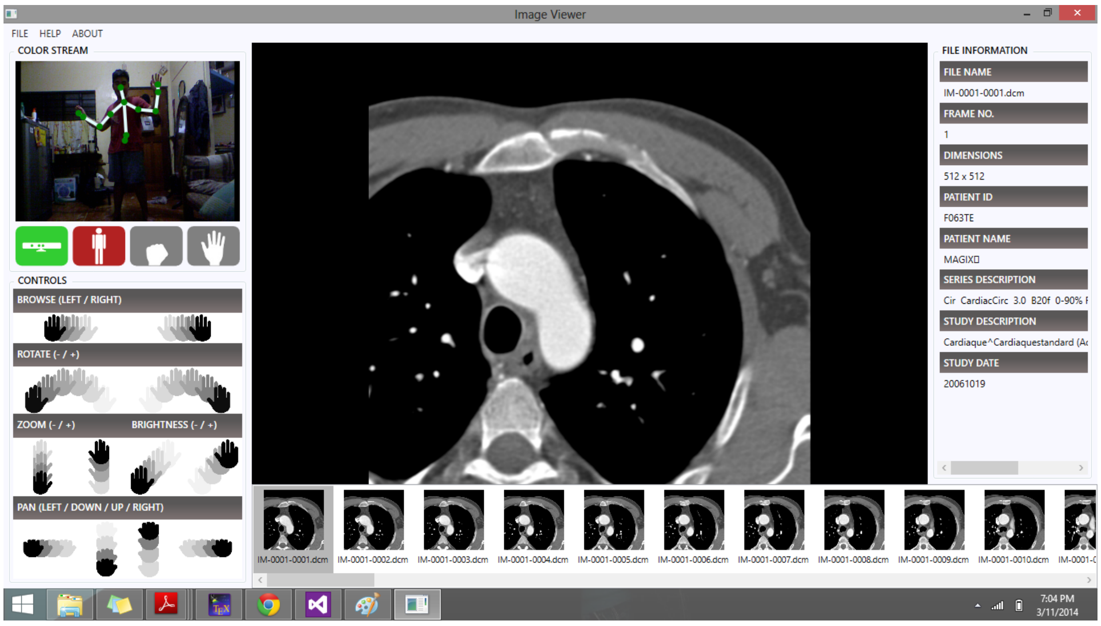
Figure 22: User not registered



FILE NAME
IM-0001-0001.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIXO
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 23: Wave gesture

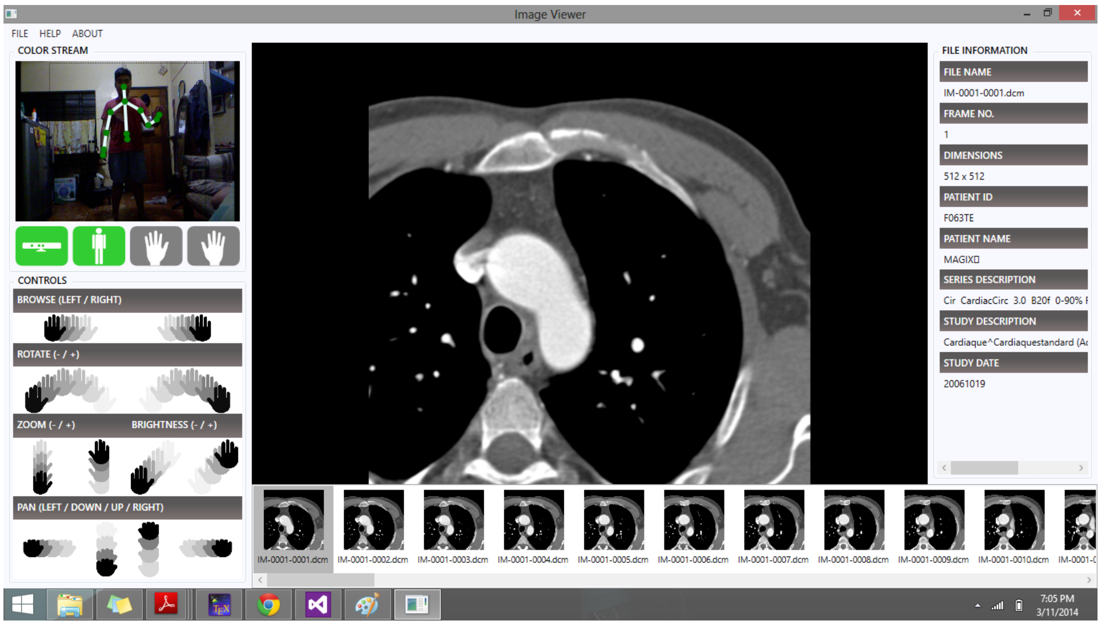
When the user status indicator turns yellow, it means that a potential user is visible in the video stream (Figure 22). If the potential user performs a waving hand gesture, he/she will be tracked and will now be able to perform image navigation and manipulation (Figure 11). Other potential users will be disregarded.



FILE INFORMATION

FILE NAME	IM-0001-0001.dcm
FRAME NO.	1
DIMENSIONS	512 x 512
PATIENT ID	F063TE
PATIENT NAME	MAGIX
SERIES DESCRIPTION	Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION	Cardiaque^Cardiaquestandard (A
STUDY DATE	20061019

Figure 24: User unstable

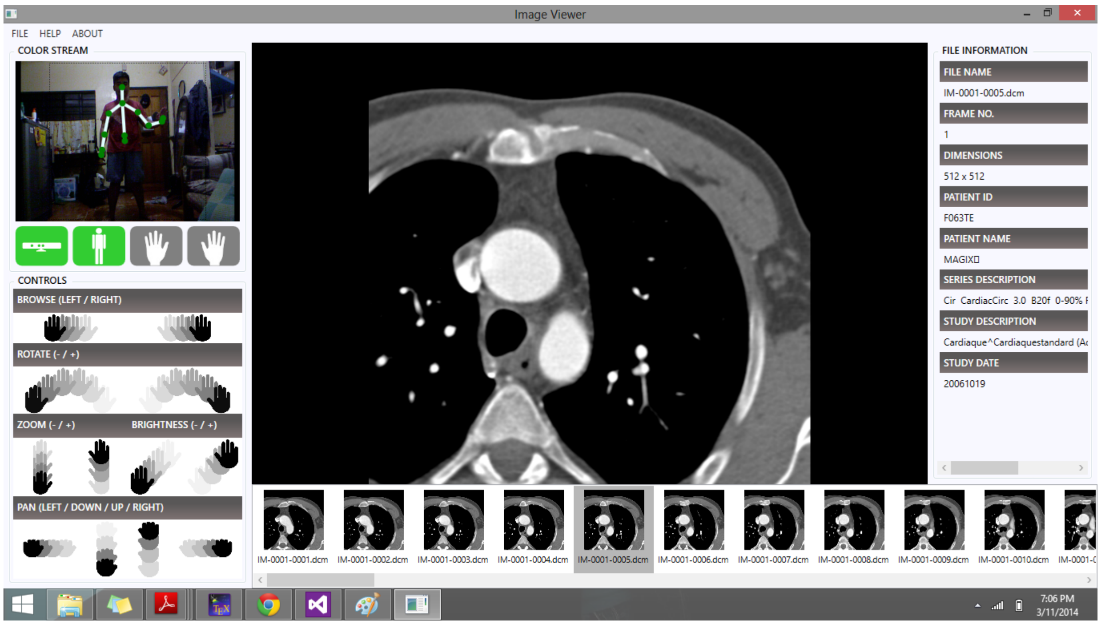


FILE INFORMATION

FILE NAME	IM-0001-0001.dcm
FRAME NO.	1
DIMENSIONS	512 x 512
PATIENT ID	F063TE
PATIENT NAME	MAGIX
SERIES DESCRIPTION	Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION	Cardiaque^Cardiaquestandard (A
STUDY DATE	20061019

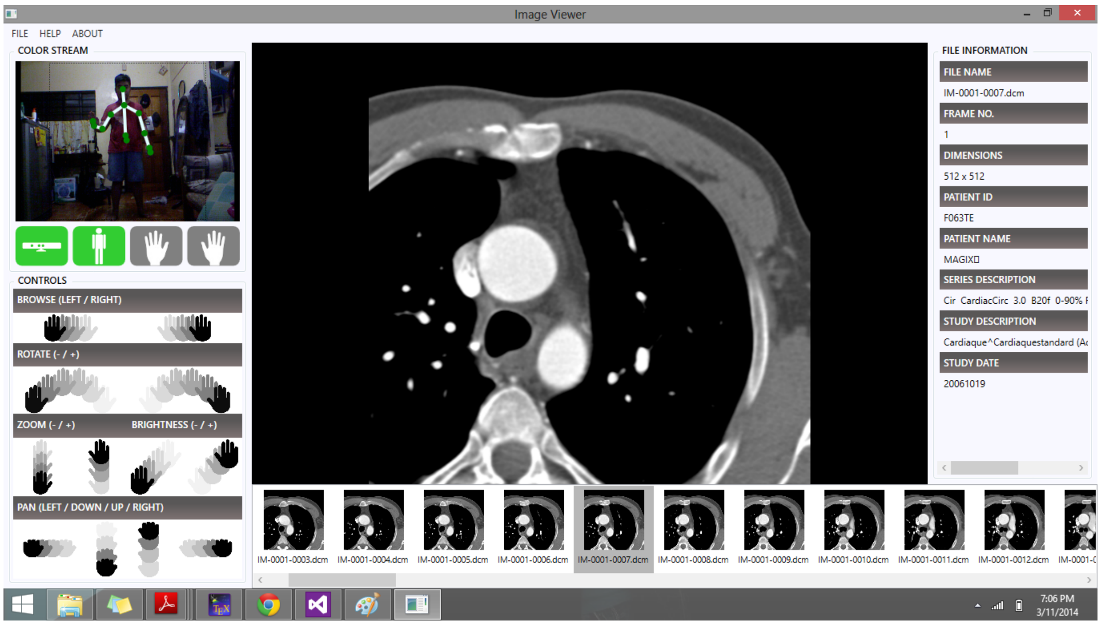
Figure 25: User stable

If the user is not facing towards the sensor or if his/her hands are both up or both down, the user becomes unstable and the user status indicator will turn red (Figure 24). This means that any movement done by the user will be ignored thus, no gestures will be recognized. If one of the hands are raised, the user becomes stable and the indicator will turn green (Figure 25). The user can now perform gestures in this position.



FILE INFORMATION
FILE NAME
IM-0001-0005.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIXD
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 26: Browse right gesture

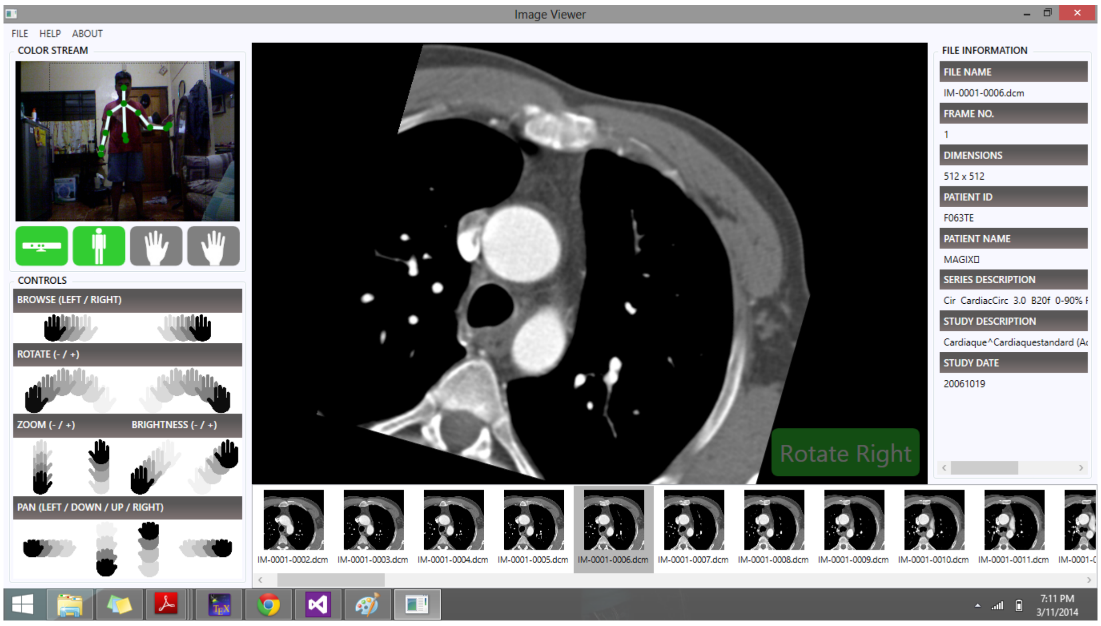


FILE INFORMATION
FILE NAME
IM-0001-0007.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIXD
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 27: Browse left gesture

The remaining screenshots show the different gestures the user can perform in this application and their corresponding commands.

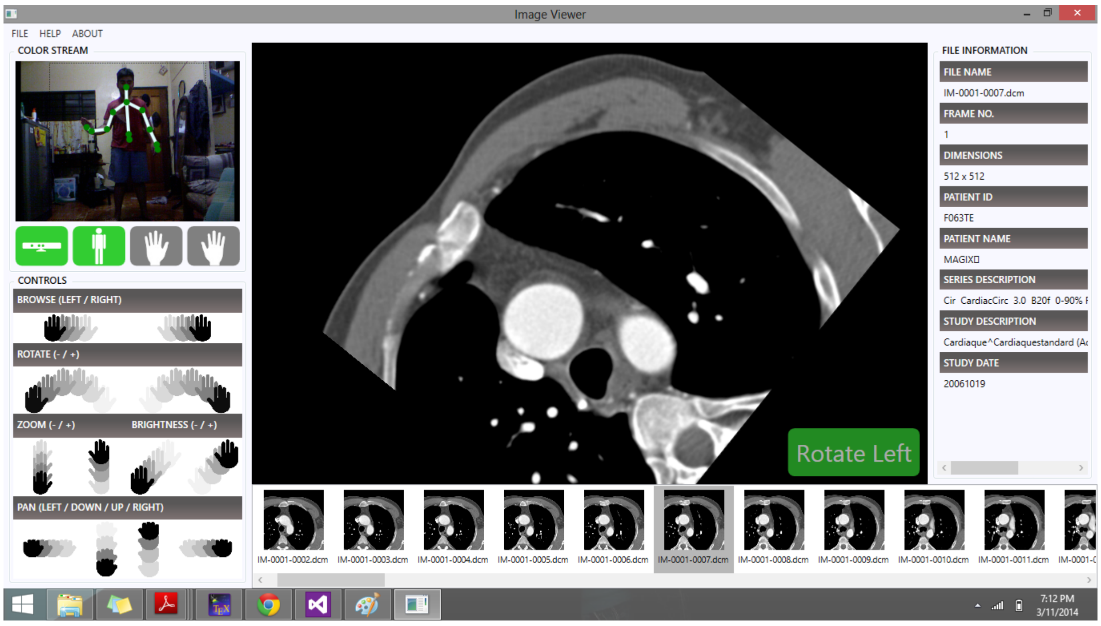
In figures 26 and 27, the user performs the browse right and browse left gestures.



FILE INFORMATION

FILE NAME
IM-0001-0006.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIX□
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 28: Rotate right gesture

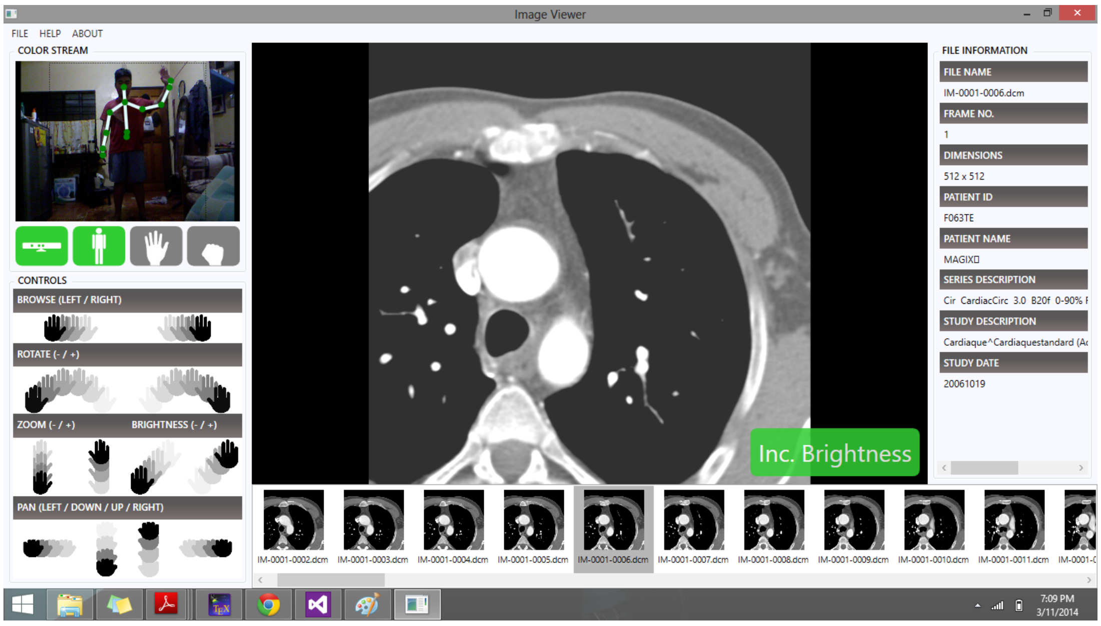


FILE INFORMATION

FILE NAME
IM-0001-0007.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIX□
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 29: Rotate left gesture

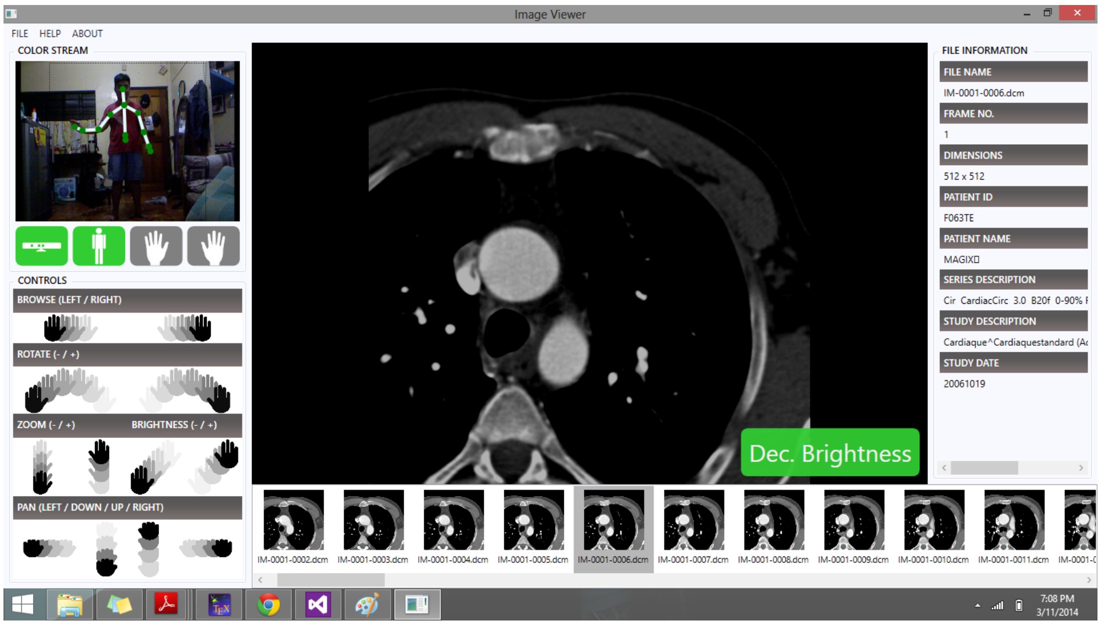
The rotate right and rotate left gestures are demonstrated in Figures 28 and 29.



FILE INFORMATION

FILE NAME
IM-0001-0006.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIXO
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 30: Brightness up gesture

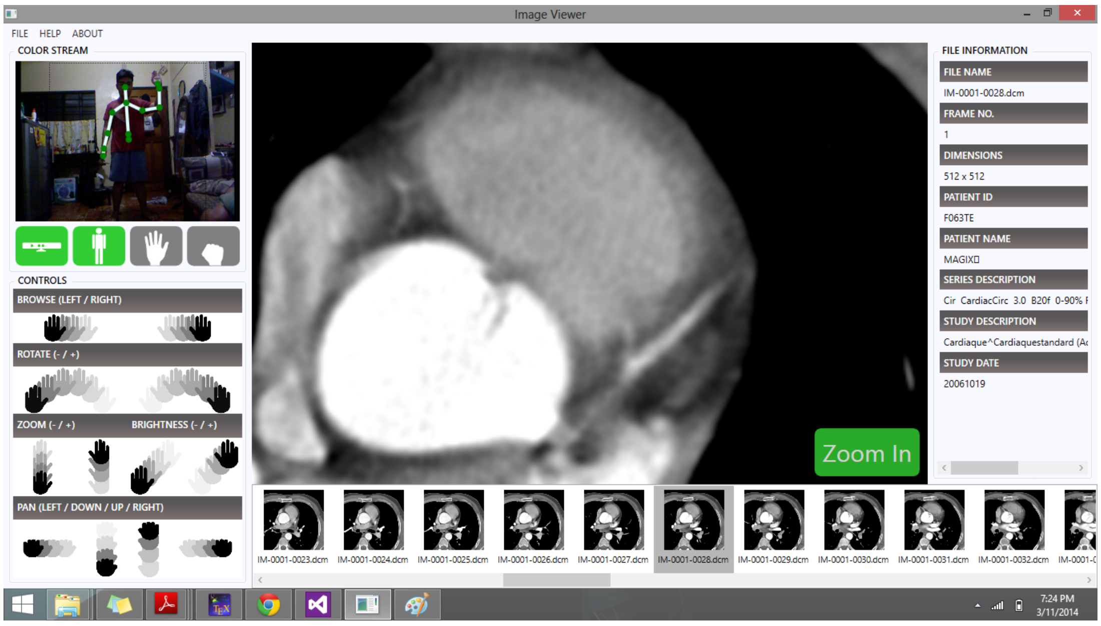


FILE INFORMATION

FILE NAME
IM-0001-0006.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIXO
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 31: Brightness down gesture

In Figures 30 and 31, the user performs diagonal swipe gestures that increase and decrease the brightness of the image.



FILE INFORMATION

FILE NAME
IM-0001-0028.dcm

FRAME NO.
1

DIMENSIONS
512 x 512

PATIENT ID
F063TE

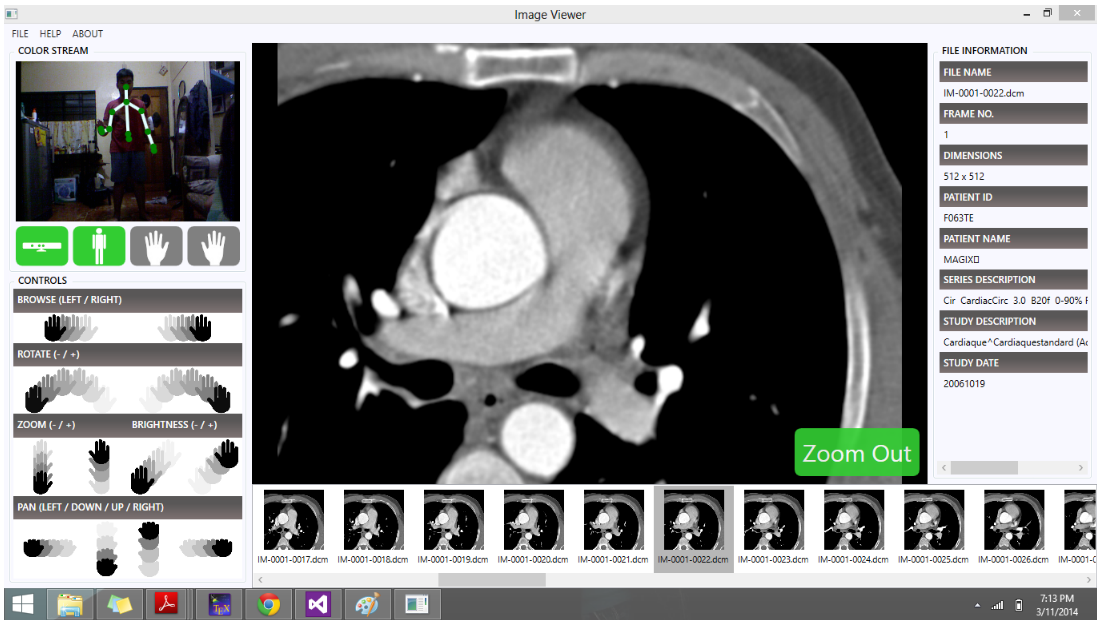
PATIENT NAME
MAGIXO

SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%

STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A

STUDY DATE
20061019

Figure 32: Zoom in gesture



FILE INFORMATION

FILE NAME
IM-0001-0022.dcm

FRAME NO.
1

DIMENSIONS
512 x 512

PATIENT ID
F063TE

PATIENT NAME
MAGIXO

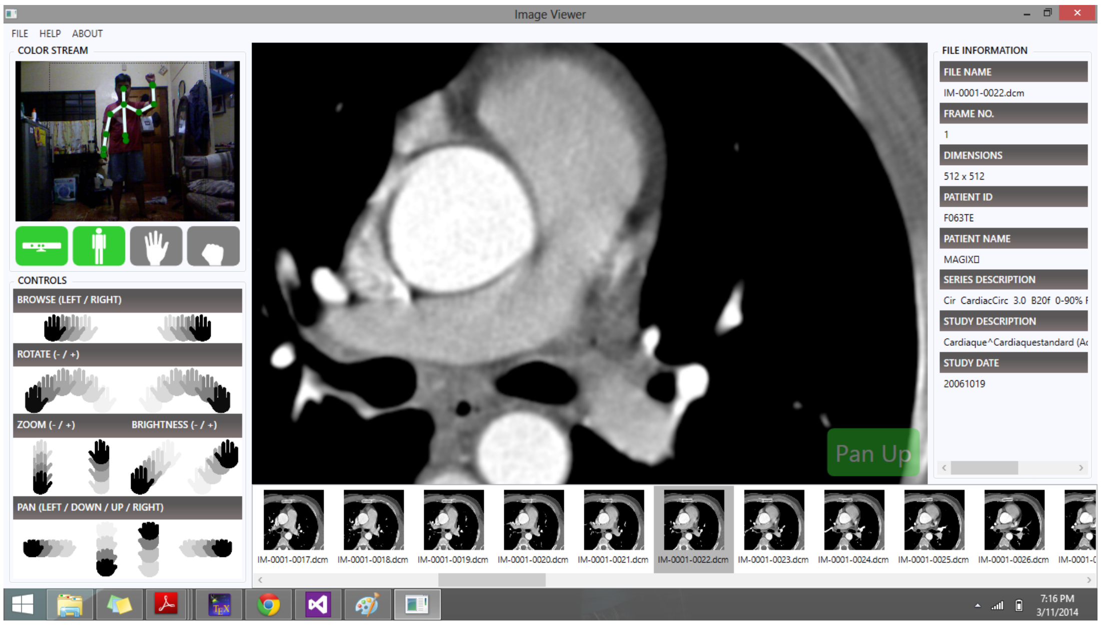
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%

STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A

STUDY DATE
20061019

Figure 33: Zoom out gesture

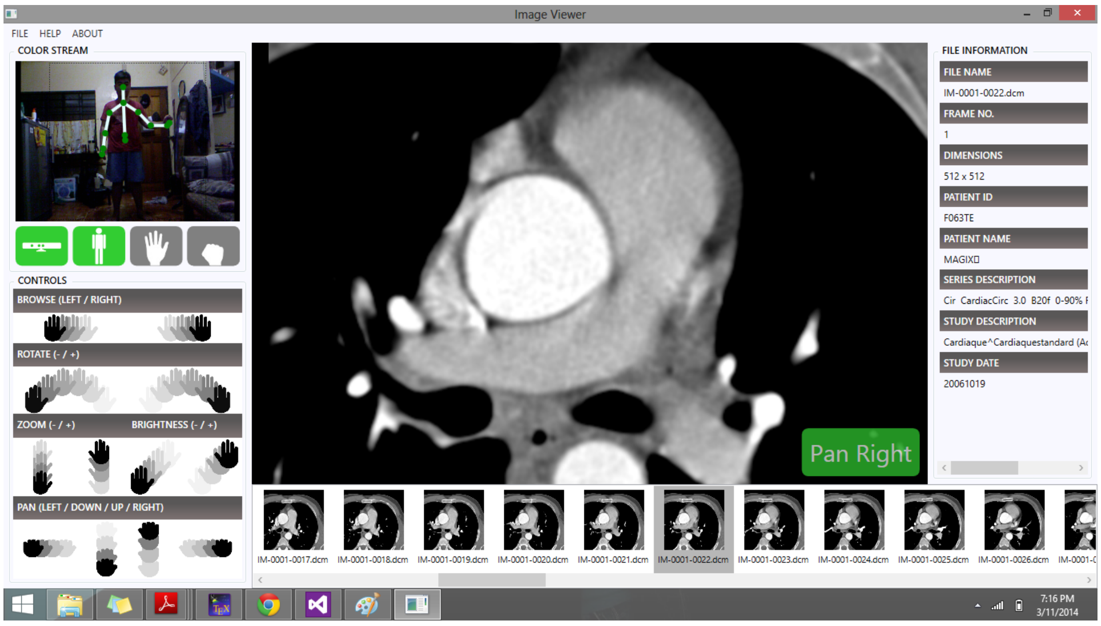
The user performs zoom in and zoom out gestures in Figures 32 and 33.



FILE INFORMATION

FILE NAME
IM-0001-0022.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIX
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 34: Pan up gesture

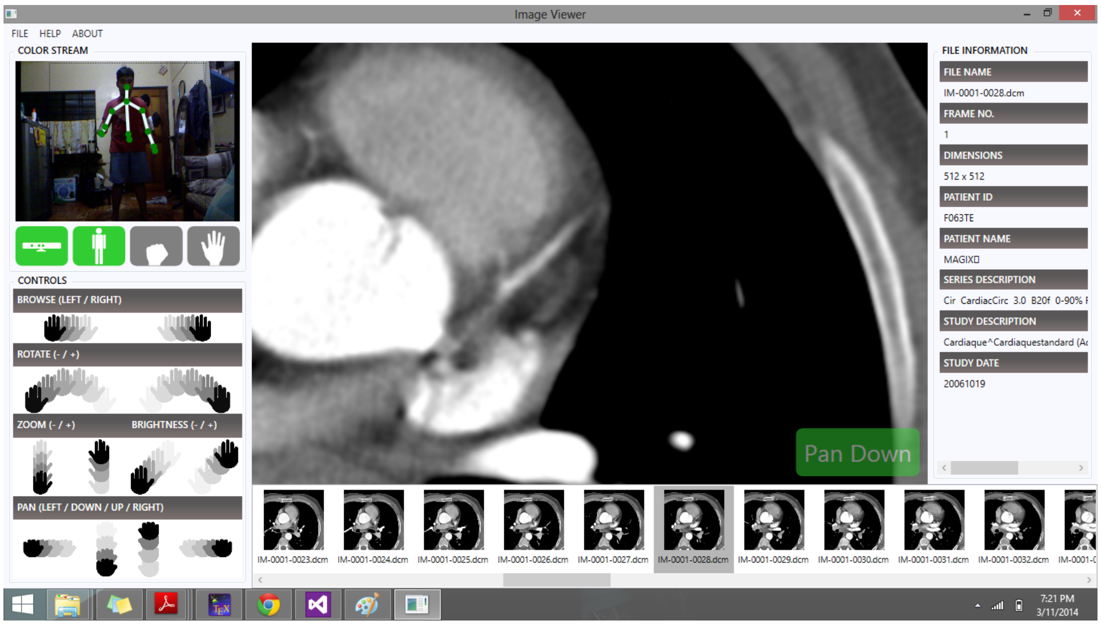


FILE INFORMATION

FILE NAME
IM-0001-0022.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIX
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (A
STUDY DATE
20061019

Figure 35: Pan right gesture

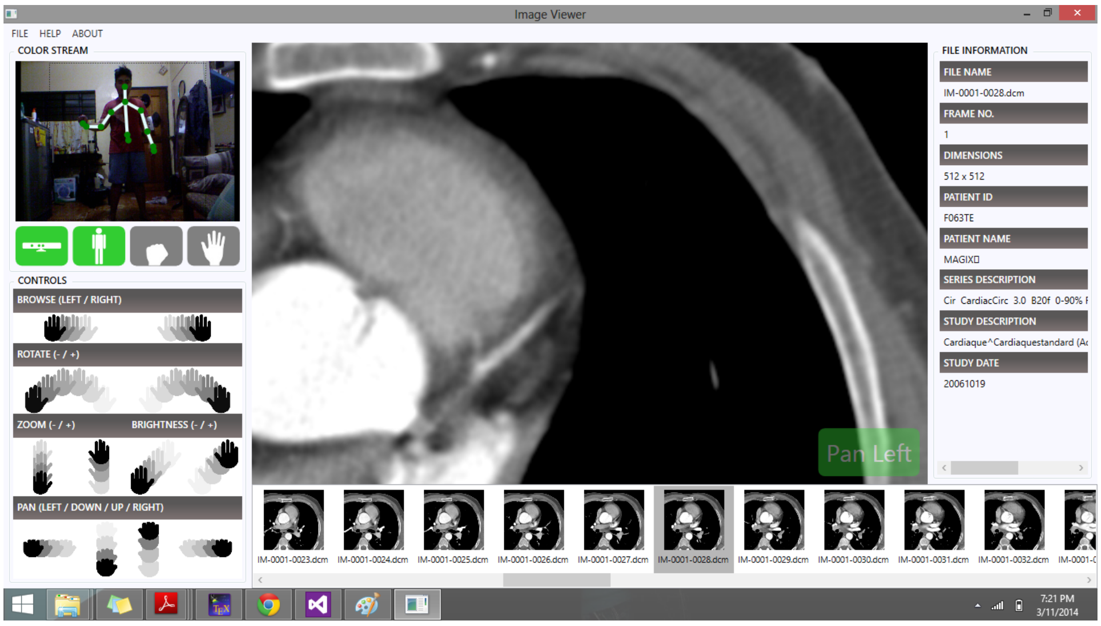
When the image is zoomed in, it becomes too big to fit in the Image Display View panel, so the user can perform panning to show the unseen parts of the image. The pan up and pan right gestures are shown in Figures 34 and 35.



FILE INFORMATION

FILE NAME	IM-0001-0028.dcm
FRAME NO.	1
DIMENSIONS	512 x 512
PATIENT ID	F063TE
PATIENT NAME	MAGIXD
SERIES DESCRIPTION	Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION	Cardiaque^Cardiaquestandard (A
STUDY DATE	20061019

Figure 36: Pan down gesture

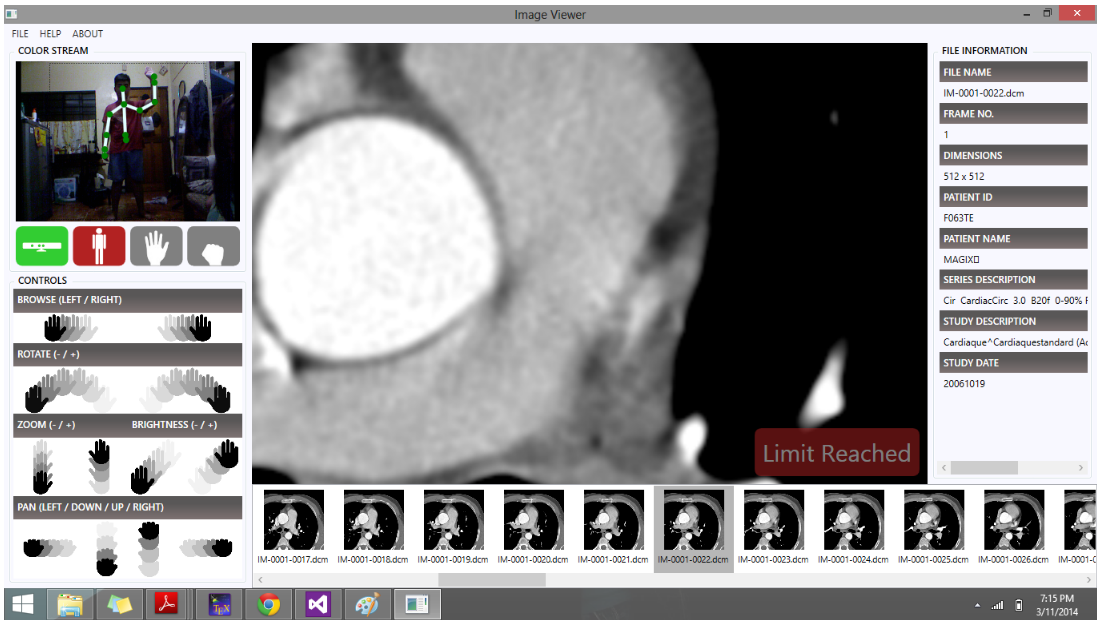


FILE INFORMATION

FILE NAME	IM-0001-0028.dcm
FRAME NO.	1
DIMENSIONS	512 x 512
PATIENT ID	F063TE
PATIENT NAME	MAGIXD
SERIES DESCRIPTION	Cir CardiacCirc 3.0 B20f 0-90%
STUDY DESCRIPTION	Cardiaque^Cardiaquestandard (A
STUDY DATE	20061019

Figure 37: Pan left gesture

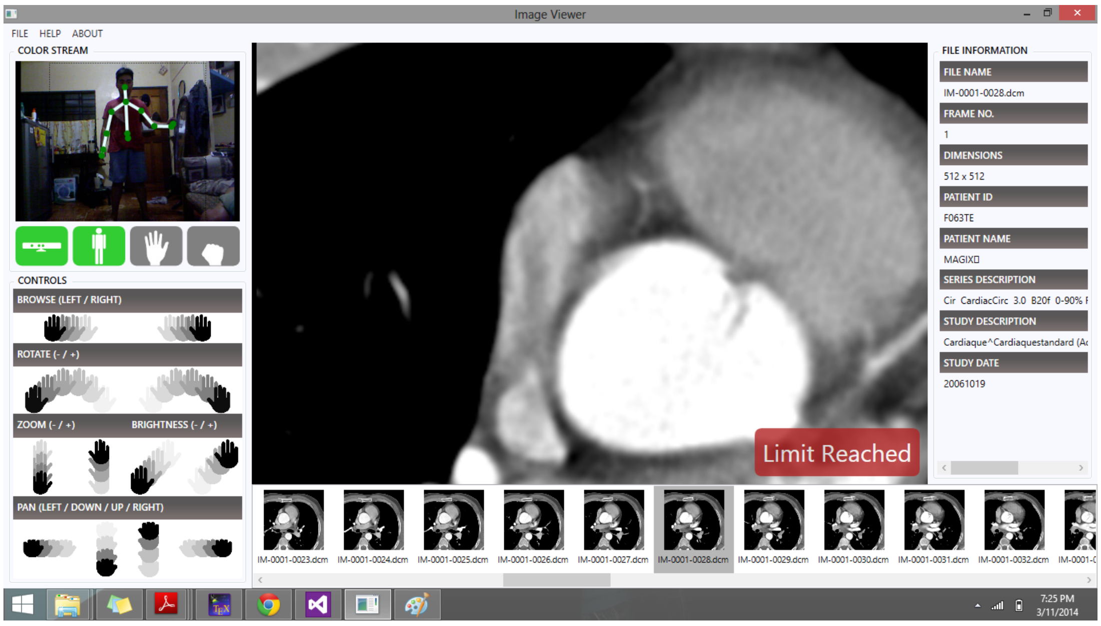
Lastly, the pan down and pan left gestures are shown in Figures 36 and 37.



FILE INFORMATION

FILE NAME
IM-0001-0022.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIX□
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90% F
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (Ad
STUDY DATE
20061019

Figure 38: Zoom limit reached



FILE INFORMATION

FILE NAME
IM-0001-0028.dcm
FRAME NO.
1
DIMENSIONS
512 x 512
PATIENT ID
F063TE
PATIENT NAME
MAGIX□
SERIES DESCRIPTION
Cir CardiacCirc 3.0 B20f 0-90% F
STUDY DESCRIPTION
Cardiaque^Cardiaquestandard (Ad
STUDY DATE
20061019

Figure 39: Pan limit reached

There will be a limit in terms of zooming and panning. So when a user tries to zoom and pan too much, the gestures will still be recognized but there will be no effect on the image (Figures 38 and 39).

VI. Discussion

The gesture-controlled image navigation and manipulation application presented in this paper is a Kinect-based application that the surgeon can use whenever there is a need to manipulate or navigate through medical images during surgical operations. Image navigation and manipulation is performed through gestural commands so that the surgeon does not need to use the mouse or keyboard thereby maintaining sterility. The application recognizes gestural commands for browsing, rotating, zooming, panning and adjusting the brightness of medical images.

The Kinect sensor along with the Kinect for Windows SDK provides a way for developers to easily apply features necessary for developing Kinect-based applications, from video streaming to user detection and skeletal tracking. However, it also has its downsides. First, user detection through skeletal tracking requires the user's full body. If only the upper half of the user's body is visible, the sensor might not detect the user properly. The sensor also works poorly when there is too much sunlight in the environment. It is also advisable to have a clean environment because the sensor can sometimes detect random objects in the viewable area as users.

The movements for each gesture are made to be not too complex so that the gestures are easy to learn and can be performed with minimum effort. At the same time, the gestures are evenly distributed between the left and right hands so as to prevent false detection and to reduce the chance of recognizing unintentional gestures. The choice of which gestures to put in the gesture vocabulary matters because it also affects how well the gesture recognition engine recognizes and differentiates intentional from unintentional gestures.

The threshold values and the delay between gestures are important factors that can also affect the performance of the gesture recognition engine. Setting a high threshold value may trigger false detection but setting it very low may result to intentional but undetected gestures. On the other hand, the setting of delay between gestures depends on the command the gesture represents. For example, the browse gesture requires shorter delays so that there will be a continuous movement to the next or previous images. Whereas the gestures for zooming and rotating require longer delays to prevent overlapping of commands and to make time for the animation.

Algorithmic and template-based gesture recognition were the two approaches used in this application. Although the algorithmic approach would suffice given the types of gestures used, the template-based approach simplifies the process of checking the constraints and conditions for each gesture.

There are still other approaches that can be used for gesture recognition. For example, in [9, 13, 14], the Hidden Markov Model (HMM), which is based on Bayesian networks was used. The template-based approach was also applied in [21] but with the Golden Section Search as its pattern matching algorithm. Besides hand gesture recognition, finger recognition was also implemented in [13, 14]. In this application however, only a closed and an open hand can be recognized since the Kinect for Windows SDK does not currently support finger tracking. But developers can use the depth data provided by the SDK to be able to implement it.

This application was developed using C#. The Windows Presentation Foundation (WPF), a graphical subsystem for rendering interfaces is used in creating the application's user interface. WPF uses Model-View-ViewModel (MVVM), an architectural pattern which facilitates the separation of the user interface code from the business logic. The API used for the rendering of DICOM images is Fellow Oak DICOM (fo-dicom) [24].

VII. Conclusion

The main purpose of this project is to enable the surgeon to perform navigation and manipulation of medical images during surgery through the use of hand gestures. Hand gesture recognition is a type of human-computer interaction that does not require a mouse, keyboard or any other controller which makes it ideal to implement in the operating room, where physical contact with objects is not allowed in order to maintain sterility and prevent the spread of infections.

The gesture-controlled application presented in this paper made use of the computer vision capabilities of the Kinect sensor in order to implement user detection and hand gesture recognition. The application allows the user to load an image or a set of images and perform commands for navigation and manipulation. The gestural commands that can be performed include browsing to the next/previous images, zooming in/out, panning left/right/up/down, rotating 90° left/right and increasing/decreasing brightness. By checking the user's stability and whether the user performed a wave gesture, the user's intention to use the application can be determined.

The application only made use of the skeletal tracking feature of the Kinect for Windows SDK, but there are still other interesting features which can be implemented alongside skeletal tracking in order to develop more sophisticated gesture-controlled applications. Other features like depth data, infrared data and even face tracking can be used on future projects.

VIII. Recommendation

Besides skeletal tracking, the Kinect for Windows SDK provides even more features and development tools that can enable the developer to create better gesture-controlled applications. In terms of providing a more efficient gesture recognition engine for this application, one can implement finger tracking and recognition using the depth data provided by the SDK. A gesture recognition engine that supports finger tracking and recognition requires lesser arm movement which can result to easier navigation and manipulation of images. Making slight changes to threshold and gesture delay values can also contribute to the improvement of the performance of the gesture recognition engine. The face tracking feature of the SDK can be used to check whether the user is facing the Kinect sensor or to enable face recognition of users. However, face tracking is computationally intensive; implementing it might slow down the application.

In terms of DICOM image visualization, one can implement DICOM surface rendering to enable 3D viewing and then add the necessary gestural commands for the manipulation of 3D images [25]. Unfortunately, the Fellow Oak DICOM API, which was used in this application only supports 2D image rendering; there may be a need to use a different API in order to view DICOM images in 3D.

IX. Bibliography

[1] A. Jana, *Kinect for Windows SDK Programming Guide.* Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing Ltd., 2012.

[2] C. Graetzel, T. Fong, S. Grange, and C. Baur, "A non-contact mouse for surgeon-computer interaction," *Technology and Health Care*, 2003.

[3] D. N. A. Tagoe and F. Kumi, "Computer keyboard and mice: Potential sources of disease transmission and infections," *The Internet Journal of Public Health*, 2011.

[4] American Society of Anesthesiologists, *Operating Room Design Manual*, 2010–2011.

[5] J. Wachs, H. Stern, Y. Edan, M. Gillam, C. Feied, M. Smith, and J. Handler, "Gestix: A doctor-computer sterile gesture interface for dynamic environments," *Soft Computing in Industrial Applications*, pp. 30–39, 2007.

[6] P. Halarnkar, S. Shah, H. Shah, H. Shah, and J. Shah, "Gesture recognition technology: A review," *International Journal of Engineering Science and Technology (IJEST)*, pp. 4648–4654, 2012.

[7] B. A. Myers, "A brief history of human computer interaction technology," *ACM Interactions*, pp. 44–54, 1998.

[8] P. Garg, N. Aggarwal, and S. Sofat, "Vision based hand gesture recognition," *World Academy of Science, Engineering and Technology 25 2009*, pp. 972–977, 2009.

[9] D. L. Achacon, Jr., D. Carlos, and M. K. Puyaoan, "REALISM: Real-time hand gesture interface for surgeons and medical experts," *Proceedings of the Philippine Computing Science Congress*, 2009.

[10] R. A. El-laithy, J. Huang, and M. Yeh, "Study on the use of microsoft kinect for robotics applications," *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pp. 1280–1288, 2012.

[11] M. A. M. Shukran and M. S. B. Ariffin, "Kinect-based gesture password recognition," *Australian Journal of Basic and Applied Sciences*, pp. 492–499, 2012.

[12] F. Soltani, F. Eskandari, and S. Golestan, "Developing a gesture-based game for deaf/mute people using microsoft kinect," *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, 2012.

[13] M. Jacob, C. Cange, R. Parker, and J. P. Wachs, "Intention, context and gesture recognition for sterile mri navigation in the operating room," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 220–227, 2012.

[14] M. G. Jacob and J. P. Wachs, "Context-based hand gesture recognition for the operating room," *Pattern Recognition Letters*, 2013.

[15] D. Gafurov and E. Snekkkenes, "A fast algorithm for vision-based hand gesture recognition for robot control," *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IIHMSP '08 International Conference on*, pp. 1080–1087, 2008.

[16] J. Guerra-Casanova, C. S. Avila, G. Bailador, and A. de Santos Sierra, "Authentication in mobile devices through hand gesture recognition," *International Journal of Information Security*, pp. 65–83, 2012.

[17] C.-C. Wang and K.-C. Wang, "Hand posture recognition using adaboost with sift for human robot interaction," *Recent Progress in Robotics: Viable Robotic Service to Human*, pp. 317–329, 2008.

[18] A. Malima, O. E., and M. Cetin, “Arm swing as a weak biometric for unobtrusive user authentication,” *Signal Processing and Communications Applications, 2006 IEEE 14th*, pp. 1–4, 2006.

[19] T. Starner and A. Pentland, “Real-time american sign language recognition from video using hidden markov models,” *Proceedings of International Symposium on Computer Vision*, pp. 265–270, 1995.

[20] J. Smisek, M. Jancosek, and T. Pajdla, “3d with kinect,” *2011 IEEE International Conference on Computer Vision Workshops*, 2011.

[21] D. Catuhe, *Programming with the Kinect for Windows Software Development Kit.* Redmond, Washington 98052-6399: Microsoft Press, 2012.

[22] U. of Reading, “Probabilistic neural network (pnn).” `http://www.personal.reading.ac.uk/~sis01xh/teaching/CY2D2/Pattern3.pdf`. [Online; accessed 11-March-2014].

[23] U. of Illinois, “Gaussian classifiers.” `http://luthuli.cs.uiuc.edu/~daf/courses/CS-498-DAF-PS/Lecture%206%20-%20Gaussian%20Classifiers.pdf`. [Online; accessed 11-March-2014].

[24] H. Desouky, “Fellow oak dicom for .net.” `https://github.com/rcd/fo-dicom`. [Online; accessed 22-December-2014].

[25] B. Kugle, “Medical image visualization using wpf.” `http://www.codeproject.com/Articles/466955/Medical-image-visualization-using-WPF`. [Online; accessed 25-March-2014].

X. Appendix

App.xaml

```
<Application x:Class="ImageViewer2.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:vm="clr-namespace:ImageViewer2"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <vm:SeriesInfoViewModel x:Key="viewModel"/>
        <vm:ImageListViewModel x:Key="viewModel2"/>
        <vm:SelectedImageViewModel x:Key="viewModel3"/>

        <vm:KinectViewModel x:Key="kinectViewModel"/>
        <vm:ImageProcessesViewModel x:Key="imageProcessesViewModel"/>
    </Application.Resources>
</Application>
```

App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace ImageViewer2
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
    }
}
```

MainWindow.xaml

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:effect="clr-namespace:BrightnessEffect;assembly=BrightnessEffect"
    x:Name="mainWindowCtrl" x:Class="ImageViewer2.MainWindow"
    Title="Image Viewer" WindowState="Maximized"
    Closed="mainWindowCtrl_Closed"
    Closing="mainWindowCtrl_Closing">
    <!-- RESOURCES PART -->
    <Window.Resources>
        <!-- CONVERTERS -->
        <BooleanToVisibilityConverter x:Key="VisConverter" />
        <!-- STYLES -->
        <ControlTemplate x:Key="kinectButtonTemp" TargetType="{x:Type Button}">
            <Grid>
                <Ellipse x:Name="outerCircle" Width="100" Height="100">
                    <Ellipse.Fill>
                        <LinearGradientBrush EndPoint="0.5,1" MappingMode="RelativeToBoundingBox"
                              StartPoint="0.5,0">
                            <GradientStop Color="#FF595757" Offset="0.296"/>
                            <GradientStop Color="#FF7E7373" Offset="1"/>
                        </LinearGradientBrush>
                    </Ellipse.Fill>
                </Ellipse>
                <Ellipse x:Name="innerCircle" Fill="GhostWhite" Width="80" Height="80"/>
                <Viewbox>
                    <ContentControl Margin="3" Content="{TemplateBinding Content}"/>
                </Viewbox>
            </Grid>
            <ControlTemplate.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter TargetName="outerCircle" Property="Fill" Value="ForestGreen" />
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
        <Style x:Key="HeaderStyle" TargetType="{x:Type Label}">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1" MappingMode="RelativeToBoundingBox" StartPoint="0.5,0">
                        <GradientStop Color="#FF595757" Offset="0.296"/>
                        <GradientStop Color="#FF7E7373" Offset="1"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Style>
        <Style x:Key="ListboxItemStyle" TargetType="{x:Type ListBoxItem}">
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="ListBoxItem">
                        <Border Name="borderCtrl" SnapsToDevicePixels="True">
                            <ContentPresenter />
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsSelected" Value="True">
                                <Setter TargetName="borderCtrl" Property="Background" Value="#B7B7B7" />
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
        <!-- KINECT ICON STATUS -->
        <Style x:Key="Animations5" TargetType="{x:Type Button}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding HideKinectIcon, Source={StaticResource kinectViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="Opacity"
                                    From="1.0" To="0.0" Duration="0:0:1"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
            </Style.Triggers>
        </Style>
        <!-- IMAGE ANIMATION STATUS -->
        <Style x:Key="Animations3" TargetType="{x:Type Border}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding ActionAnimation, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="Opacity"
                                    From="0.0" To="1.0" Duration="0:0:1" AutoReverse="True"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
            </Style.Triggers>
        </Style>
        <Style x:Key="Animations4" TargetType="{x:Type Border}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding LimitAnimation, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="Opacity"
                                    From="0.0" To="1.0" Duration="0:0:1" AutoReverse="True"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
            </Style.Triggers>
        </Style>
        <!-- IMAGE ANIMATIONS -->
        <Style x:Key="Animations2" TargetType="{x:Type Grid}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding ResetPos, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.X"
                                    To="0" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2" />
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Y"
                                    To="0" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2" />
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding PanUp, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Y"
                                        By="-150" Duration="0:0:1" AccelerationRatio="0.8"
                                             DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding PanLeft, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.X"
                                        By="-150" Duration="0:0:1" AccelerationRatio="0.8"
                                             DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding PanRight, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.X"
                                        By="150" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding PanDown, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Y"
                                        By="150" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
            </Style.Triggers>
        </Style>
        <Style x:Key="Animations" TargetType="{x:Type Image}">
            <Style.Triggers>
                <DataTrigger Binding="{Binding RotateRight, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Children[1].Angle"
                                        By="90" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding RotateLeft, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Children[1].Angle"
                                        By="-90" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding ZoomIn, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Children[0].ScaleX"
                                        By="0.4" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Children[0].ScaleY"
                                        By="0.4" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding ZoomOut, Source={StaticResource imageProcessesViewModel}}"
                     Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Children[0].ScaleX"
                                        By="-0.4" Duration="0:0:1" AccelerationRatio="0.8"
                                             DecelerationRatio="0.2"/>
                                <DoubleAnimation Storyboard.TargetProperty="RenderTransform.Children[0].ScaleY"
                                        By="-0.4" Duration="0:0:1" AccelerationRatio="0.8"
                                             DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding IncreaseBrightness, Source={StaticResource
                     imageProcessesViewModel}}" Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation
                                     Storyboard.TargetProperty="(UIElement.Effect).(effect:BrightnessEffect.Brightness)"
                                        By="0.1" Duration="0:0:1" AccelerationRatio="0.8" DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
                <DataTrigger Binding="{Binding DecreaseBrightness, Source={StaticResource
                     imageProcessesViewModel}}" Value="True">
                    <DataTrigger.EnterActions>
                        <BeginStoryboard HandoffBehavior="Compose">
                            <Storyboard>
                                <DoubleAnimation
                                     Storyboard.TargetProperty="(UIElement.Effect).(effect:BrightnessEffect.Brightness)"
                                        By="-0.1" Duration="0:0:1" AccelerationRatio="0.8"
                                             DecelerationRatio="0.2"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </DataTrigger.EnterActions>
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>
    <!-- USER INTERFACE -->
    <DockPanel
```

```
        HorizontalAlignment="Stretch"
        LastChildFill="True"
        VerticalAlignment="Stretch"
Background="GhostWhite"
        DataContext="{Binding Source={StaticResource viewModel3}}">
<!-- MENU BAR -->
<Menu Margin="3 3 3 3" DockPanel.Dock="Top" IsMainMenu="True" Background="GhostWhite">
    <MenuItem Header="_FILE">
        <MenuItem x:Name="loadImageCtrl" Header="_Load Image File..." Click="loadImageCtrl_Click"/>
        <MenuItem x:Name="loadSeriesCtrl" Header="_Load Series..." Click="loadSeriesCtrl_Click"/>
        <Separator/>
        <MenuItem x:Name="exitCtrl" Header="_Exit"/>
    </MenuItem>
    <MenuItem Header="_HELP"/>
    <MenuItem Header="_ABOUT"/>
</Menu>
<!-- KINECT DATA PANEL -->
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="0.73*" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <!-- COLOR STREAM -->
    <GroupBox Margin="6 0 6 0">
        <GroupBox.Header>
            <TextBlock Margin="3 0 3 3" Text="COLOR STREAM" FontWeight="SemiBold" />
        </GroupBox.Header>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <Grid Margin="3" Background="Black" Width="280" Height="200">
                <Image Source="{Binding Bitmap, Source={StaticResource kinectViewModel}}"/>
                <ItemsControl ItemsSource="{Binding Bones, Source={StaticResource kinectViewModel}}">
                    <ItemsControl.ItemsPanel>
                        <ItemsPanelTemplate>
                            <Canvas IsItemsHost="True"/>
                        </ItemsPanelTemplate>
                    </ItemsControl.ItemsPanel>
                    <ItemsControl.ItemTemplate>
                        <DataTemplate>
                            <Line X1="{Binding VectorOne.X}" Y1="{Binding VectorOne.Y}"
                                                                   X2="{Binding VectorTwo.X}" Y2="{Binding
                                                                        VectorTwo.Y}"
                                                                   Stroke="LightCyan" StrokeThickness="4"/>
                        </DataTemplate>
                    </ItemsControl.ItemTemplate>
                </ItemsControl>
                <ItemsControl ItemsSource="{Binding Joints, Source={StaticResource kinectViewModel}}">
                    <ItemsControl.ItemsPanel>
                        <ItemsPanelTemplate>
                            <Canvas IsItemsHost="True"/>
                        </ItemsPanelTemplate>
                    </ItemsControl.ItemsPanel>
                    <ItemsControl.ItemContainerStyle>
                        <Style>
                            <Setter Property="Canvas.Left" Value="{Binding X}"/>
                            <Setter Property="Canvas.Top" Value="{Binding Y}"/>
                        </Style>
                    </ItemsControl.ItemContainerStyle>
                    <ItemsControl.ItemTemplate>
                        <DataTemplate>
                            <Ellipse Fill="Green" Width="8" Height="8">
                                <Ellipse.RenderTransform>
                                    <TranslateTransform X="-4" Y="-4"/>
                                </Ellipse.RenderTransform>
                            </Ellipse>
                        </DataTemplate>
                    </ItemsControl.ItemTemplate>
                </ItemsControl>
                <Button Grid.Row="1" x:Name="initKinectCtrl"
                        HorizontalAlignment="Center"
                        VerticalAlignment="Center"
                        Opacity="1.0"
                        Style="{StaticResource Animations5}"
                        Template="{StaticResource kinectButtonTemp}"
                        Click="initKinectCtrl_Click">
                    <Button.Content>
                        <Image Source="Images.KinectLogo/kinectlogo.png" Height="3" Width="3" />
                    </Button.Content>
                </Button>
            </Grid>
            <!-- USER STATUS -->
            <Grid Grid.Row="1">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <!-- KINECT STATUS -->
                <Border Margin="3" CornerRadius="8,8,8,8" Background="{Binding KinectStatusColor,
                     Source={StaticResource kinectViewModel}}">
                    <Image Source="Images.Hand/kinect2.png" />
                </Border>
                <!-- BODY STATUS -->
                <Border Margin="3" Grid.Column="1" CornerRadius="8,8,8,8" Background="{Binding
                     UserStatusColor, Source={StaticResource kinectViewModel}}" Padding="5 2 5 2">
```

```
                        <Image Source="Images.Hand/userfinal2.png" />
                    </Border>
                    <!-- HAND STATUS -->
                    <Border Margin="3" Grid.Column="2" CornerRadius="8,8,8,8" Background="Gray">
                        <Image Source="Images.Hand/openhand2.png" />
                    </Border>
                    <Border Margin="3" Grid.Column="2" CornerRadius="8,8,8,8" Background="Gray"
                        Visibility="{Binding LeftHand, Source={StaticResource kinectViewModel},
                             Converter={StaticResource VisConverter}}">
                        <Image Source="Images.Hand/closedhand2.png" />
                    </Border>
                    <Border Margin="3" Grid.Column="3" CornerRadius="8,8,8,8" Background="Gray">
                        <Image Source="Images.Hand/openhand.png" />
                    </Border>
                    <Border Margin="3" Grid.Column="3" CornerRadius="8,8,8,8" Background="Gray"
                        Visibility="{Binding RightHand, Source={StaticResource kinectViewModel},
                             Converter={StaticResource VisConverter}}">
                        <Image Source="Images.Hand/closedhand.png" />
                    </Border>
                </Grid>
            </Grid>
        </GroupBox>
        <!-- CONTROLS DISPLAY -->
        <GroupBox Margin="6 0 6 6" Grid.Row="1">
            <GroupBox.Header>
                <TextBlock Margin="3 0 3 3" Text="CONTROLS" FontWeight="SemiBold" />
            </GroupBox.Header>
            <Grid Background="White">
                <Grid.RowDefinitions>
                    <RowDefinition Height="*" />
                    <RowDefinition Height="1.3*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="2*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="2.5*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="2.5*" />
                </Grid.RowDefinitions>
                <Label Grid.Row="0" Content="BROWSE (LEFT / RIGHT)" Foreground="White" FontWeight="SemiBold"
                     Style="{StaticResource HeaderStyle}" />
                <Grid Grid.Row="1" Margin="1">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Image Grid.Column="0" Source="Images.Gestures/open_left.png" />
                    <Image Grid.Column="1" Source="Images.Gestures/open_right.png" />
                </Grid>
                <Label Grid.Row="2" Content="ROTATE (- / +)" Foreground="White" FontWeight="SemiBold"
                     Style="{StaticResource HeaderStyle}" />
                <Grid Grid.Row="3" Margin="1">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Image Grid.Column="0" Source="Images.Gestures/arc_left.jpg" />
                    <Image Grid.Column="1" Source="Images.Gestures/arc_right.jpg" />
                </Grid>
                <Grid Grid.Row="4">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Label Grid.Column="0" Content="ZOOM (- / +)" Foreground="White" FontWeight="SemiBold"
                         Style="{StaticResource HeaderStyle}" />
                    <Label Grid.Column="1" Content="BRIGHTNESS (- / +)" Foreground="White"
                         FontWeight="SemiBold" Style="{StaticResource HeaderStyle}" />
                </Grid>
                <Grid Grid.Row="5" Margin="1">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Image Grid.Column="0" Source="Images.Gestures/open_down.png" />
                    <Image Grid.Column="1" Source="Images.Gestures/open_up.png" />
                    <Image Grid.Column="2" Source="Images.Gestures/diagonal_down.png" />
                    <Image Grid.Column="3" Source="Images.Gestures/diagonal_up.png" />
                </Grid>
                <Label Grid.Row="6" Content="PAN (LEFT / DOWN / UP / RIGHT)" Foreground="White"
                     FontWeight="SemiBold" Style="{StaticResource HeaderStyle}" />
                <Grid Grid.Row="7" Margin="1">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="*" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <Image Grid.Column="0" Source="Images.Gestures/pan1.png" />
                    <Image Grid.Column="1" Source="Images.Gestures/pan2.png" />
                </Grid>
            </Grid>
        </GroupBox>
    </Grid>
    <!-- IMAGE FRAMES -->
    <ListBox x:Name="imageListCtrl"
                DockPanel.Dock="Bottom"
                Height="130"
                ScrollViewer.HorizontalScrollBarVisibility="Visible"
                ItemsSource="{Binding ImageCollection, Source={StaticResource viewModel2}}"
```

```
                    SelectedIndex="{Binding SelectedIndex, Source={StaticResource viewModel2}}"
        ItemContainerStyle="{StaticResource ListboxItemStyle}"
                    SelectionChanged="imageListCtrl_SelectionChanged">
        <ListBox.ItemsPanel>
            <ItemsPanelTemplate>
                <StackPanel Orientation="Horizontal"/>
            </ItemsPanelTemplate>
        </ListBox.ItemsPanel>
        <ListBox.ItemTemplate>
            <DataTemplate>
                <Grid Width="100" Height="100">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="3*"/>
                        <RowDefinition Height="*"/>
                    </Grid.RowDefinitions>
                    <Image Source="{Binding Image}"/>
                    <Label Grid.Row="1" Content="{Binding ImageInfo.FileName}" FontSize="10.5" />
                </Grid>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
    <!-- OPTIONS AND METADATA PANEL -->
    <GroupBox Margin="6 0 6 6" DockPanel.Dock="Right" Width="200">
        <GroupBox.Header>
            <TextBlock Margin="3 0 3 3" Text="FILE INFORMATION" FontWeight="SemiBold" />
        </GroupBox.Header>
        <ScrollViewer HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility="Auto">
            <StackPanel Margin="3">
                <Label Content="FILE NAME" Foreground="White" FontWeight="SemiBold" Style="{StaticResource
                     HeaderStyle}"/>
                <Label x:Name="fileNameCtrl" Content="{Binding SelectedImage.ImageInfo.FileName}"/>

                <Label Content="FRAME NO." Foreground="White" FontWeight="SemiBold" Style="{StaticResource
                     HeaderStyle}"/>
                <Label x:Name="frameNoCtrl" Content="{Binding SelectedImage.ImageInfo.FrameNo}"/>

                <Label Content="DIMENSIONS" Foreground="White" FontWeight="SemiBold" Style="{StaticResource
                     HeaderStyle}"/>
                <Label x:Name="dimensionsCtrl" Content="{Binding SelectedImage.ImageInfo.Dimensions}"/>

                <Label Content="PATIENT ID" Foreground="White" FontWeight="SemiBold" Style="{StaticResource
                     HeaderStyle}"/>
                <Label x:Name="patientIDCtrl" Content="{Binding SelectedImage.ImageInfo.PatientID}"/>

                <Label Content="PATIENT NAME" Foreground="White" FontWeight="SemiBold" Style="{StaticResource
                     HeaderStyle}"/>
                <Label x:Name="patientNameCtrl" Content="{Binding SelectedImage.ImageInfo.PatientName}"/>

                <Label Content="SERIES DESCRIPTION" Foreground="White" FontWeight="SemiBold"
                     Style="{StaticResource HeaderStyle}"/>
                <Label x:Name="seriesDescCtrl" Content="{Binding SelectedImage.ImageInfo.SeriesDesc}"/>

                <Label Content="STUDY DESCRIPTION" Foreground="White" FontWeight="SemiBold"
                     Style="{StaticResource HeaderStyle}"/>
                <Label x:Name="studyDescCtrl" Content="{Binding SelectedImage.ImageInfo.StudyDesc}"/>

                <Label Content="STUDY DATE" Foreground="White" FontWeight="SemiBold" Style="{StaticResource
                     HeaderStyle}"/>
                <Label x:Name="studyDateCtrl" Content="{Binding SelectedImage.ImageInfo.StudyDate}"/>
            </StackPanel>
        </ScrollViewer>
    </GroupBox>
    <!-- MAIN IMAGE VIEW -->
    <Grid>
        <Border x:Name="mainImageBorderCtrl" ClipToBounds="True" Background="Black">
            <Grid x:Name="imageGridCtrl" Style="{StaticResource Animations2}">
                <Grid.RenderTransform>
                    <TranslateTransform Changed="TranslateTransform_Changed"/>
                </Grid.RenderTransform>
                <Image x:Name="mainImageCtrl" RenderTransformOrigin="0.5, 0.5"
                       Source="{Binding SelectedImage.Image}" Style="{StaticResource Animations}">
                    <Image.RenderTransform>
                        <TransformGroup>
                            <ScaleTransform Changed="ScaleTransform_Changed"/>
                            <RotateTransform />
                        </TransformGroup>
                    </Image.RenderTransform>
                    <Image.Effect>
                        <effect:BrightnessEffect/>
                    </Image.Effect>
                </Image>
            </Grid>
        </Border>
        <Border CornerRadius="8,8,8,8" Background="LimeGreen"
                Padding="10" Margin="10" Opacity="0.0"
                VerticalAlignment="Bottom" HorizontalAlignment="Right"
                Style="{StaticResource Animations3}">
            <TextBlock Foreground="White" FontSize="30"
                       Text="{Binding ActionStatus, Source={StaticResource imageProcessesViewModel}}"/>
        </Border>
        <Border CornerRadius="8,8,8,8" Background="Firebrick"
                Padding="10" Margin="10" Opacity="0.0"
                VerticalAlignment="Bottom" HorizontalAlignment="Right"
                Style="{StaticResource Animations4}">
            <TextBlock Foreground="White" FontSize="30" Text="Limit Reached"/>
        </Border>
    </Grid>
</DockPanel>
</Window>
```

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using Microsoft.Kinect;

namespace ImageViewer2
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private ImageListViewModel imageListVM;
        private SelectedImageViewModel selectedImageVM;
        private KinectViewModel kinectServiceVM;
        private ImageProcessesViewModel imageProcessVM;

        private int prevIndex = 0;
        private int zoomCounter = 0;

        private Rect rect;
        private Rect bounds;
        private Rect container;

        public MainWindow()
        {
            InitializeComponent();

            imageListVM = (ImageListViewModel)FindResource("viewModel2");
            selectedImageVM = (SelectedImageViewModel)FindResource("viewModel3");

            kinectServiceVM = (KinectViewModel)FindResource("kinectViewModel");
            imageProcessVM = (ImageProcessesViewModel)FindResource("imageProcessesViewModel");
        }

        private void loadSeriesCtrl_Click(object sender, RoutedEventArgs e)
        {
            ChooseSeriesWindow seriesChooser = new ChooseSeriesWindow();
            seriesChooser.Show();
        }

        private void mainWindowCtrl_Closed(object sender, EventArgs e)
        {
            for (int i = App.Current.Windows.Count - 1; i >= 0; i--)
                App.Current.Windows[i].Close();
        }

        private void imageListCtrl_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            selectedImageVM.UpdateSelectedImage((MedicalImage)imageListCtrl.SelectedItem);

            if (prevIndex == imageListCtrl.SelectedIndex + 1)
            {
                if (imageListCtrl.SelectedIndex - 4 >= 0)
                    imageListCtrl.ScrollIntoView(imageListCtrl.Items[imageListCtrl.SelectedIndex - 4]);
            }
            else if(prevIndex == imageListCtrl.SelectedIndex - 1)
            {
                if(imageListCtrl.SelectedIndex + 4 < imageListCtrl.Items.Count)
                    imageListCtrl.ScrollIntoView(imageListCtrl.Items[imageListCtrl.SelectedIndex + 4]);
            }

            prevIndex = imageListCtrl.SelectedIndex;
        }

        private void loadImageCtrl_Click(object sender, RoutedEventArgs e)
        {
            imageListVM.UpdateImageList();
        }

        private void mainWindowCtrl_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            kinectServiceVM.CleanUpResources();
        }

        private void initKinectCtrl_Click(object sender, RoutedEventArgs e)
        {
            rect = new Rect(imageGridCtrl.RenderSize);
            bounds = imageGridCtrl.RenderTransform.TransformBounds(rect);
            container = new Rect(new Size(mainImageBorderCtrl.ActualWidth, mainImageBorderCtrl.ActualHeight));
            kinectServiceVM.ImageRenderBounds = bounds;
            kinectServiceVM.ImageContainerBounds = container;

            kinectServiceVM.StartKinectSensor();
        }
        private void TranslateTransform_Changed(object sender, EventArgs e)
        {
            rect = new Rect(mainImageCtrl.RenderSize);
            bounds = mainImageCtrl.TransformToAncestor(mainImageBorderCtrl).TransformBounds(rect);
            kinectServiceVM.ImageRenderBounds = bounds;
        }

        private void ScaleTransform_Changed(object sender, EventArgs e)
        {
            rect = new Rect(mainImageCtrl.RenderSize);
            bounds = mainImageCtrl.TransformToAncestor(mainImageBorderCtrl).TransformBounds(rect);
            kinectServiceVM.ImageRenderBounds = bounds;
        }
    }
}
```

ChooseSeriesWindow.xaml

```
<Window x:Name="chooseSeriesWindowCtrl"
        x:Class="ImageViewer2.ChooseSeriesWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Select Series"
        Height="310" Width="520"
        WindowStartupLocation="CenterScreen"
        ResizeMode="NoResize"
        ShowInTaskbar="False">
    <Grid Background="GhostWhite" Margin="6">
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="5*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="4*" />
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <TextBlock HorizontalAlignment="Left" VerticalAlignment="Center"
                       TextWrapping="WrapWithOverflow"
                       Margin="6"
                       Text="Browse for the folder containing the files then select a series below." />
            <Button Grid.Column="1" x:Name="browseButtonCtrl" Margin="6"  Content="Browse..."
                 Click="browseButtonCtrl_Click"/>
        </Grid>
        <DataGrid Grid.Row="1" Margin="6" x:Name="seriesGridCtrl"
                  SelectionMode="Single"
                  AutoGenerateColumns="False" IsReadOnly="True"
                  CanUserReorderColumns="False" CanUserSortColumns="False"
                  CanUserResizeColumns="False"
                  HorizontalGridLinesBrush="DarkGray"
                  VerticalGridLinesBrush="DarkGray"
                  HorizontalScrollBarVisibility="Visible"
                  ItemsSource="{Binding Source={StaticResource viewModel}, Path=SeriesInfoCollection}">
            <DataGrid.Resources>
                <Style TargetType="DataGridCell">
                    <Setter Property="BorderThickness" Value="0"/>
                    <Setter Property="FocusVisualStyle" Value="{x:Null}"/>
                </Style>
            </DataGrid.Resources>
            <DataGrid.Columns>
                <DataGridTextColumn Width="*" Header="Patient ID" Binding="{Binding PatientID}" />
                <DataGridTextColumn Width="*" Header="Patient Name" Binding="{Binding PatientName}" />
                <DataGridTextColumn Width="2*" Header="Series Description" Binding="{Binding SeriesDesc}" />
                <DataGridTextColumn Width="*" Header="Series Date" Binding="{Binding SeriesDate}" />
            </DataGrid.Columns>
        </DataGrid>
        <Grid Grid.Row="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="3*" />
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
            <Button Grid.Column="1" x:Name="selectButtonCtrl" Margin="6" Content="OK"
                 Click="selectButtonCtrl_Click"/>
            <Button Grid.Column="2" x:Name="cancelButtonCtrl" Margin="6" Content="Cancel"
                 Click="cancelButtonCtrl_Click"/>
        </Grid>
    </Grid>
</Window>
```

ChooseSeriesWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace ImageViewer2
{
    /// <summary>
    /// Interaction logic for ChooseSeriesWindow.xaml
    /// </summary>
    public partial class ChooseSeriesWindow : Window
    {
        private SeriesInfoViewModel seriesInfoVM;
        private ImageListViewModel imageListVM;

        public ChooseSeriesWindow()
        {
            InitializeComponent();

            seriesInfoVM = (SeriesInfoViewModel)FindResource("viewModel");
            imageListVM = (ImageListViewModel)FindResource("viewModel2");
        }

        private void browseButtonCtrl_Click(object sender, RoutedEventArgs e)
        {
            seriesInfoVM.GetSeriesList();
        }

        private void cancelButtonCtrl_Click(object sender, RoutedEventArgs e)
        {
            seriesInfoVM.ClearSeriesList();
            this.Close();
        }

        private void selectButtonCtrl_Click(object sender, RoutedEventArgs e)
        {
            imageListVM.UpdateImageList(seriesInfoVM.SelectedPath, (SeriesInfo)seriesGridCtrl.SelectedItem);
            seriesInfoVM.ClearSeriesList();
            this.Close();
        }
    }
}
```

MedicalImage.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Drawing;
using System.Windows.Media.Imaging;
using Dicom;
using Dicom.Imaging;
using System.IO;

namespace ImageViewer2
{
    public class MedicalImage
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="path"></param>
        /// <param name="img"></param>
        /// <param name="isDicom"></param>
        public MedicalImage(string path, Image img, bool isDicom)
        {
            Image = ConvertToBitmapImage(new Bitmap(img));
            ImageInfo = new MedicalImageInfo(path, isDicom);
        }

        #endregion

        #region Members

        private BitmapImage _image;
        private MedicalImageInfo _imageInfo;

        #endregion

        #region Properties

        /// <summary>
        /// The medical image
        /// </summary>
        public BitmapImage Image
        {
            get { return _image; }
            set { _image = value; }
        }

        /// <summary>
        /// The information regarding the medical image
        /// </summary>
        public MedicalImageInfo ImageInfo
        {
            get { return _imageInfo; }
            private set { _imageInfo = value; }
        }

        #endregion

        #region ConvertToBitmapImage() method
        /// <summary>
        /// Converts a Bitmap object to a BitmapImage object
        /// </summary>
        /// <param name="img"></param>
        /// <returns></returns>
        private BitmapImage ConvertToBitmapImage(Bitmap img)
        {
            BitmapImage bitmapImg = new BitmapImage();

            using (MemoryStream ms2 = new MemoryStream())
            {
                img.Save(ms2, System.Drawing.Imaging.ImageFormat.Bmp);
                ms2.Position = 0;
                ms2.Seek(0, SeekOrigin.Begin);
                bitmapImg.BeginInit();
                bitmapImg.StreamSource = ms2;
                bitmapImg.CacheOption = BitmapCacheOption.OnLoad;
                bitmapImg.EndInit();

                return bitmapImg;
            }
        }

        #endregion
    }
}
```

MedicalImageInfo.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Dicom;
using System.Windows.Media.Imaging;

namespace ImageViewer2
{
    public class MedicalImageInfo
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="path"></param>
        /// <param name="isDicom"></param>
        public MedicalImageInfo(string path, bool isDicom)
        {
            FileName = Path.GetFileName(path);
            FrameNo = 1; // default frame number

            if (isDicom)
            {
                DicomFile dcmFile = DicomFile.Open(path);

                if (dcmFile.Dataset.Get<string>(DicomTag.PatientID) != null)
                    PatientID = dcmFile.Dataset.Get<string>(DicomTag.PatientID);

                if (dcmFile.Dataset.Get<string>(DicomTag.PatientName) != null)
                    PatientName = dcmFile.Dataset.Get<string>(DicomTag.PatientName);

                if (dcmFile.Dataset.Get<string>(DicomTag.StudyDescription) != null)
                    StudyDesc = dcmFile.Dataset.Get<string>(DicomTag.StudyDescription);

                if (dcmFile.Dataset.Get<string>(DicomTag.StudyDate) != null)
                    StudyDate = dcmFile.Dataset.Get<string>(DicomTag.StudyDate);

                if (dcmFile.Dataset.Get<string>(DicomTag.SeriesDescription) != null)
                    SeriesDesc = dcmFile.Dataset.Get<string>(DicomTag.SeriesDescription);

                string rows = dcmFile.Dataset.Get<string>(DicomTag.Rows);
                string cols = dcmFile.Dataset.Get<string>(DicomTag.Columns);

                if(rows != null && cols != null)
                    Dimensions = rows + " x " + cols;
            }
            else
            {
                BitmapFrame frame = BitmapFrame.Create(new Uri(path), BitmapCreateOptions.DelayCreation,
                     BitmapCacheOption.None);
                _metadata = (BitmapMetadata)frame.Metadata;

                uint? width = QueryImageWidth();
                uint? height = QueryImageHeight();

                if(width != null && height != null)
                    Dimensions = width + " x " + height;
            }
        }

        #endregion

        #region Members

        private BitmapMetadata _metadata;
        private string _fileName = "Not Available";
        private string _patientID = "Not Available";
        private string _patientName = "Not Available";
        private string _studyDesc = "Not Available";
        private string _studyDate = "Not Available";
        private string _seriesDesc = "Not Available";
        private string _dimensions = "Not Available";
        private int _frameNo;

        #endregion

        #region Properties

        /// <summary>
        /// The file name of the medical image
        /// </summary>
        public string FileName
        {
            get { return _fileName; }
            private set { _fileName = value; }
        }

        /// <summary>
        /// The patient ID obtained from the DICOM tag
        /// </summary>
        public string PatientID
        {
            get { return _patientID; }
            private set { _patientID = value; }
        }

        /// <summary>
        /// The patient name obtained from the DICOM tag
        /// </summary>
        public string PatientName
        {
            get { return _patientName; }
            private set { _patientName = value; }
        }

        /// <summary>
        /// The study description obtained from the DICOM tag
        /// </summary>
        public string StudyDesc
        {
            get { return _studyDesc; }
            private set { _studyDesc = value; }
        }

        /// <summary>
        /// The study date obtained from the DICOM tag
        /// </summary>
        public string StudyDate
        {
            get { return _studyDate; }
            private set { _studyDate = value; }
        }

        /// <summary>
        /// The series description obtained from the DICOM tag
        /// </summary>
        public string SeriesDesc
        {
            get { return _seriesDesc; }
            private set { _seriesDesc = value; }
        }

        /// <summary>
        /// The dimensions of the medical image obtained from the metadata or the DICOM tag
        /// </summary>
        public string Dimensions
        {
            get { return _dimensions; }
            private set { _dimensions = value; }
        }

        /// <summary>
        /// The frame number of the image obtained from the DICOM tag
        /// </summary>
        public int FrameNo
        {
            get { return _frameNo; }
            set { _frameNo = value; }
        }

        #endregion

        #region QueryMetadata() method

        /// <summary>
        /// Obtains the metadata of the current medical image
        /// </summary>
        /// <param name="query"></param>
        /// <returns></returns>
        private object QueryMetadata(string query)
        {
            if (_metadata.ContainsQuery(query))
                return _metadata.GetQuery(query);
```

```
            else
                return null;
        }

        #endregion

        #region QueryImageWidth() method

        /// <summary>
        /// Gets the image width
        /// </summary>
        /// <returns></returns>
        private uint? QueryImageWidth()
        {
            object val = QueryMetadata("/app1/ifd/exif/subifd:{uint=40962}");
            if (val == null)
            {
                return null;
            }
            else
            {
                if (val.GetType() == typeof(UInt32))
                {
                    return (uint?)val;
                }
                else
                {
                    return System.Convert.ToUInt32(val);
                }
            }
        }

        #endregion

        #region QueryImageHeight

        /// <summary>
        /// Gets the image height
        /// </summary>
        /// <returns></returns>
        private uint? QueryImageHeight()
        {
            object val = QueryMetadata("/app1/ifd/exif/subifd:{uint=40963}");
            if (val == null)
            {
                return null;
            }
            else
            {
                if (val.GetType() == typeof(UInt32))
                {
                    return (uint?)val;
                }
                else
                {
                    return System.Convert.ToUInt32(val);
                }
            }
        }

        #endregion
    }
}
```

SeriesInfo.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Dicom;

namespace ImageViewer2
{
    public class SeriesInfo
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="path"></param>
        public SeriesInfo(string path)
        {
            DicomFile dcmFile = DicomFile.Open(path);
            SeriesInstanceUID = dcmFile.Dataset.Get<string>(DicomTag.SeriesInstanceUID);
            PatientID = dcmFile.Dataset.Get<string>(DicomTag.PatientID);
            PatientName = dcmFile.Dataset.Get<string>(DicomTag.PatientName);
            SeriesDesc = dcmFile.Dataset.Get<string>(DicomTag.SeriesDescription);
            SeriesDate = dcmFile.Dataset.Get<string>(DicomTag.SeriesDate);
        }

        #endregion

        #region Members

        private string _seriesInstanceUID;
        private string _patientID;
        private string _patientName;
        private string _seriesDesc;
        private string _seriesDate;

        #endregion

        #region Properties

        /// <summary>
        /// The patient ID obtained from the DICOM tag
        /// </summary>
        public string PatientID
        {
            get { return _patientID; }
            private set { _patientID = value; }
        }

        /// <summary>
        /// The patient name obtained from the DICOM tag
        /// </summary>
        public string PatientName
        {
            get { return _patientName; }
            private set { _patientName = value; }
        }

        /// <summary>
        /// The series description obtained from the DICOM tag
        /// </summary>
        public string SeriesDesc
        {
            get { return _seriesDesc; }
            private set { _seriesDesc = value; }
        }

        /// <summary>
        /// The series date obtained from the DICOM tag
        /// </summary>
        public string SeriesDate
        {
            get { return _seriesDate; }
            private set { _seriesDate = value; }
        }

        /// <summary>
        /// The series instance UID obtained from the DICOM tag
        /// </summary>
        public string SeriesInstanceUID
        {
            get { return _seriesInstanceUID; }
            private set { _seriesInstanceUID = value; }
        }

        #endregion
    }
}
```

ViewModelBase.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public class ViewModelBase : INotifyPropertyChanged
    {
        /// <summary>
        /// Event handler when a property has changed
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        /// <summary>
        /// Notifies the View when properties are changed
        /// </summary>
        /// <param name="propertyName"></param>
        protected void RaisePropertyChanged(string propertyName)
        {
            PropertyChangedEventHandler handler = this.PropertyChanged;

            if (handler != null)
            {
                PropertyChangedEventArgs args = new PropertyChangedEventArgs(propertyName);
                handler(this, args);
            }
        }

        /// <summary>
        /// Notifies the View when properties are changed (overload)
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="propertyExpression"></param>
        protected void RaisePropertyChanged<T>(Expression<Func<T>> propertyExpression)
        {
            MemberExpression memberExpression = (MemberExpression)propertyExpression.Body;
            if (memberExpression == null)
                return;

            string propertyName = memberExpression.Member.Name;
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }
    }
}
```

SeriesInfoViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Windows.Forms;
using Dicom;

namespace ImageViewer2
{
    public class SeriesInfoViewModel : ViewModelBase
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        public SeriesInfoViewModel() { }

        #endregion

        #region Members

        private ObservableCollection<SeriesInfo> _seriesInfoCollection;
        private string _selectedPath;

        #endregion

        #region Properties

        /// <summary>
        /// Holds the list of the DICOM series opened by the user
        /// </summary>
        public ObservableCollection<SeriesInfo> SeriesInfoCollection
        {
            get { return _seriesInfoCollection; }
            private set
            {
                _seriesInfoCollection = value;
                RaisePropertyChanged("SeriesInfoCollection");
            }
        }

        /// <summary>
        /// The folder path that contains the DICOM series
        /// </summary>
        public string SelectedPath
        {
            get { return _selectedPath; }
            private set { _selectedPath = value; }
        }

        #endregion

        #region GetSeriesList() method

        /// <summary>
        /// Obtains the DICOM series list
        /// </summary>
        public void GetSeriesList()
        {
            ObservableCollection<SeriesInfo> sCollection = new ObservableCollection<SeriesInfo>();

            FolderBrowserDialog fbDialog = new FolderBrowserDialog();
            fbDialog.Description = "Please choose the folder containing the series of images to be studied.";
            fbDialog.ShowNewFolderButton = false;

            if (fbDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                SelectedPath = fbDialog.SelectedPath;
                DirectoryInfo directory = new DirectoryInfo(SelectedPath);
                foreach (FileInfo f in directory.GetFiles())
                {
                    try
                    {
                        SeriesInfo thisInfo = new SeriesInfo(f.FullName);
                        if (!sCollection.Any(item => item.SeriesInstanceUID == thisInfo.SeriesInstanceUID))
                            sCollection.Add(thisInfo);
                    }
                    catch (DicomFileException)
                    {
                        continue;
                    }
                }

                SeriesInfoCollection = sCollection;
            }
        }

        #endregion

        #region ClearSeriesList() method

        /// <summary>
        /// Clears the DICOM series list
        /// </summary>
        public void ClearSeriesList()
        {
            if (SeriesInfoCollection != null)
            {
                SeriesInfoCollection.Clear();
            }
        }

        #endregion
    }
}
```

ImageListViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Dicom;
using Dicom.Imaging;
using System.Drawing;
using System.Windows.Forms;

namespace ImageViewer2
{
    public class ImageListViewModel : ViewModelBase
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        public ImageListViewModel() { }

        #endregion

        #region Members

        private ObservableCollection<MedicalImage> _imageCollection;
        private int _selectedIndex;

        #endregion

        #region Properties

        /// <summary>
        /// Holds the lists of medical images
        /// </summary>
        public ObservableCollection<MedicalImage> ImageCollection
        {
            get { return _imageCollection; }
            private set
            {
                _imageCollection = value;
                RaisePropertyChanged("ImageCollection");
            }
        }

        /// <summary>
        /// The currently selected image index from the list
        /// </summary>
        public int SelectedIndex
        {
            get { return _selectedIndex; }
            set
            {
                _selectedIndex = value;
                RaisePropertyChanged("SelectedIndex");
            }
        }

        #endregion

        #region UpdateImageList() methods

        /// <summary>
        /// Reloads the content of the list of medical images
        /// </summary>
        public void UpdateImageList()
        {
            ObservableCollection<MedicalImage> miCollection = new ObservableCollection<MedicalImage>();
            int frameCtr = 1;
            OpenFileDialog ofDialog = new OpenFileDialog();
            ofDialog.Multiselect = true;

            if (ofDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                foreach (string path in ofDialog.FileNames)
                {
                    try
                    {
                        CreateDicomImage(miCollection, path, frameCtr);
                    }
                    catch (DicomFileException)
                    {
                        try
                        {
                            CreateImage(miCollection, path);
                        }
                        catch (NotSupportedException)
                        {
                            continue;
                        }
                    }
                }

                ImageCollection = miCollection;
                SelectedIndex = 0;
            }
        }

        /// <summary>
        /// Reloads the content of the list of medical images
        /// </summary>
        public void UpdateImageList(string selectedPath, SeriesInfo selectedSeries)
        {
            ObservableCollection<MedicalImage> miCollection = new ObservableCollection<MedicalImage>();
            int frameCtr = 1;

            DirectoryInfo directory = new DirectoryInfo(selectedPath);
            foreach (FileInfo f in directory.GetFiles())
            {
                try
                {
                    SeriesInfo thisInfo = new SeriesInfo(f.FullName);
                    if (selectedSeries.SeriesInstanceUID == thisInfo.SeriesInstanceUID)
                    {
                        CreateDicomImage(miCollection, f.FullName, frameCtr);
                    }
                }
                catch (DicomFileException)
                {
                    continue;
                }
            }

            ImageCollection = miCollection;
            SelectedIndex = 0;
        }

        #endregion

        #region CreateImage() method

        /// <summary>
        /// Loads the image on the list
        /// </summary>
        /// <param name="temp"></param>
        /// <param name="imagePath"></param>
        private void CreateImage(ObservableCollection<MedicalImage> temp, string imagePath)
        {
            // render to dummy Image object
            Image renderImg = Image.FromFile(imagePath);

            // create an instance of MedicalImage
            MedicalImage currImg = new MedicalImage(imagePath, renderImg, false);

            // add to temporary collection
            temp.Add(currImg);

            // dispose dummy Image object
            renderImg.Dispose();
        }

        #endregion

        #region CreateDicomImage() method

        /// <summary>
        /// Loads the DICOM image on the list
        /// </summary>
        /// <param name="temp"></param>
        /// <param name="imagePath"></param>
        /// <param name="frameCtr"></param>
        private void CreateDicomImage(ObservableCollection<MedicalImage> temp, string imagePath, int frameCtr)
        {
            // create an instance of DicomImage
            DicomImage dcmImage = new DicomImage(imagePath);

            // loop in case of multi-frame
            for (int i = 0; i < dcmImage.PixelData.NumberOfFrames; i++)
```

```
            {
                // render to dummy Image object
                Image renderImg = dcmImage.RenderImage(i);

                // create an instance of MedicalImage
                MedicalImage currImg = new MedicalImage(imagePath, renderImg, true);

                // set frame number
                currImg.ImageInfo.FrameNo = frameCtr;

                // add to temporary collection
                temp.Add(currImg);

                // dispose dummy Image object
                renderImg.Dispose();

                // increment frame number
                frameCtr++;
            }
        }

        #endregion

        #region IncrementSelectedIndex() method

        /// <summary>
        /// Moves to the next image index of the list
        /// </summary>
        public void IncrementSelectedIndex()
        {
            if (ImageCollection != null)
            {
                if ((SelectedIndex + 1) < ImageCollection.Count)
                {
                    SelectedIndex++;
                }
            }
        }

        #endregion

        #region DecrementSelectedIndex() method

        /// <summary>
        /// Moves to the previous image index of the list
        /// </summary>
        public void DecrementSelectedIndex()
        {
            if (ImageCollection != null)
            {
                if ((SelectedIndex - 1) >= 0)
                {
                    SelectedIndex--;
                }
            }
        }

        #endregion
    }
}
```

SelectedImageViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Media.Imaging;

namespace ImageViewer2
{
    public class SelectedImageViewModel : ViewModelBase
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        public SelectedImageViewModel() { }

        #endregion

        #region Members

        private MedicalImage _selectedImage;

        #endregion

        #region Properties

        /// <summary>
        /// The currently selected medical image
        /// </summary>
        public MedicalImage SelectedImage
        {
            get { return _selectedImage; }
            set
            {
                _selectedImage = value;
                RaisePropertyChanged("SelectedImage");
            }
        }

        #endregion

        #region UpdateSelectedImage() method

        /// <summary>
        /// Updates the selected image when the selected index from the image list is changed
        /// </summary>
        /// <param name="imageSelectedInList"></param>
        public void UpdateSelectedImage(MedicalImage imageSelectedInList)
        {
            SelectedImage = imageSelectedInList;
        }

        #endregion
    }
}
```

ImageProcessesViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public class ImageProcessesViewModel : ViewModelBase
    {
        public ImageProcessesViewModel() { }

        private bool _rotateRight;
        private bool _rotateLeft;
        private bool _zoomIn;
        private bool _zoomOut;
        private bool _panRight;
        private bool _panLeft;
        private bool _panUp;
        private bool _panDown;
        private bool _brightnessUp;
        private bool _brightnessDown;

        private bool _actionAnimation;
        private bool _limitAnimation;
        private string _actionStatus;

        /// <summary>
        /// Notifies the main application to start the action indicator animation
        /// </summary>
        public bool ActionAnimation
        {
            get { return _actionAnimation; }
            set
            {
                _actionAnimation = value;
                RaisePropertyChanged("ActionAnimation");
            }
        }

        /// <summary>
        /// Notifies the main application to start the limit indicator animation
        /// </summary>
        public bool LimitAnimation
        {
            get { return _limitAnimation; }
            set
            {
                _limitAnimation = value;
                RaisePropertyChanged("LimitAnimation");
            }
        }

        /// <summary>
        /// Notifies the main application to start the action status animation
        /// </summary>
        public string ActionStatus
        {
            get { return _actionStatus; }
            set
            {
                _actionStatus = value;
                RaisePropertyChanged("ActionStatus");
            }
        }

        /// <summary>
        /// Notifies the main application to start the zoom in animation
        /// </summary>
        public bool ZoomIn
        {
            get { return _zoomIn; }
            set
            {
                _zoomIn = value;
                ActionAnimation = value;
                RaisePropertyChanged("ZoomIn");
            }
        }

        /// <summary>
        /// Notifies the main application to start the zoom out animation
        /// </summary>
        public bool ZoomOut
        {
            get { return _zoomOut; }
            set
            {
                _zoomOut = value;
                ActionAnimation = value;
                RaisePropertyChanged("ZoomOut");
            }
        }

        /// <summary>
        /// Notifies the main application to start the pan right animation
        /// </summary>
        public bool PanRight
        {
            get { return _panRight; }
            set
            {
                _panRight = value;
                ActionAnimation = value;
                RaisePropertyChanged("PanRight");
            }
        }

        /// <summary>
        /// Notifies the main application to start the pan left animation
        /// </summary>
        public bool PanLeft
        {
            get { return _panLeft; }
            set
            {
                _panLeft = value;
                ActionAnimation = value;
                RaisePropertyChanged("PanLeft");
            }
        }

        /// <summary>
        /// Notifies the main application to start the pan up animation
        /// </summary>
        public bool PanUp
        {
            get { return _panUp; }
            set
            {
                _panUp = value;
                ActionAnimation = value;
                RaisePropertyChanged("PanUp");
            }
        }

        /// <summary>
        /// Notifies the main application to start the pan down animation
        /// </summary>
        public bool PanDown
        {
            get { return _panDown; }
            set
            {
                _panDown = value;
                ActionAnimation = value;
                RaisePropertyChanged("PanDown");
            }
        }

        /// <summary>
        /// Notifies the main application to start the increase brightness animation
        /// </summary>
        public bool IncreaseBrightness
        {
            get { return _brightnessUp; }
            set
            {
                _brightnessUp = value;
                ActionAnimation = value;
                RaisePropertyChanged("IncreaseBrightness");
            }
        }

        /// <summary>
        /// Notifies the main application to start the decrease brightness animation
        /// </summary>
        public bool DecreaseBrightness
        {
            get { return _brightnessDown; }
            set
            {
                _brightnessDown = value;
                ActionAnimation = value;
                RaisePropertyChanged("DecreaseBrightness");
            }
        }

        /// <summary>
        /// Notifies the main application to start the rotate right animation
        /// </summary>
        public bool RotateRight
        {
            get { return _rotateRight; }
            set
            {
                _rotateRight = value;
                ActionAnimation = value;
                RaisePropertyChanged("RotateRight");
            }
        }

        /// <summary>
        /// Notifies the main application to start the rotate left animation
        /// </summary>
        public bool RotateLeft
        {
            get { return _rotateLeft; }
            set
            {
                _rotateLeft = value;
                ActionAnimation = value;
                RaisePropertyChanged("RotateLeft");
            }
        }
    }
}
```

KinectViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect;
using System.Windows;
using System.Windows.Controls;
using System.Collections.ObjectModel;

using Microsoft.Kinect.Toolkit.Interaction;
using System.Windows.Media.Imaging;

namespace ImageViewer2
{
    public partial class KinectViewModel : ViewModelBase
    {
        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        public KinectViewModel() { }

        #endregion

        #region Members and Properties

        private KinectSensor _sensor;
        private InteractionStream _interactionStream;
        private StabilityChecker _stabilityChecker;

        private GestureDetector _gestureDetector;
        private WaveGestureDetector _waveDetector;

        private UserInfo[] userInfoList;
        private Skeleton[] skeletonList;

        private int _currentTrackingId = 0;
        private Skeleton _skeleton = null;
        private Skeleton _skeleton2 = null;

        private bool rightHandClosed = false;
        private bool leftHandClosed = false;
        private int _zoomCounter = 0;

        /// <summary>
        /// Status of the left hand
        /// </summary>
        private bool _leftHand;
        public bool LeftHand
        {
            get { return _leftHand; }
            set
            {
                _leftHand = value;
                RaisePropertyChanged("LeftHand");
            }
        }

        /// <summary>
        /// Status of the right hand
        /// </summary>
        private bool _rightHand;
```

```
public bool RightHand
{
    get { return _rightHand; }
    set
    {
        _rightHand = value;
        RaisePropertyChanged("RightHand");
    }
}

/// <summary>
/// Notifies the View to hide the Kinect button
/// </summary>
private bool _hideKinectIcon;
public bool HideKinectIcon
{
    get { return _hideKinectIcon; }
    set
    {
        _hideKinectIcon = value;
        RaisePropertyChanged("HideKinectIcon");
    }
}

/// <summary>
/// Notifies the View of the status of the Kinect
/// </summary>
private string _kinectStatusColor = "Gray";
public string KinectStatusColor
{
    get { return _kinectStatusColor; }
    set
    {
        _kinectStatusColor = value;
        RaisePropertyChanged("KinectStatusColor");
    }
}

/// <summary>
/// Notifies the View of the status of the user
/// </summary>
private string _userStatusColor = "Gray";
public string UserStatusColor
{
    get { return _userStatusColor; }
    set
    {
        _userStatusColor = value;
        RaisePropertyChanged("UserStatusColor");
    }
}

/// <summary>
/// The bounds of the currently selected image
/// </summary>
private Rect _imageRenderBounds;
public Rect ImageRenderBounds
{
    get { return _imageRenderBounds; }
    set
    {
        _imageRenderBounds = value;
        RaisePropertyChanged("ImageRenderBounds");
    }
}

/// <summary>
/// The bounds of the container of the currently selected image
/// </summary>
private Rect _imageContainerBounds;
public Rect ImageContainerBounds
{
    get { return _imageContainerBounds; }
    set
    {
        _imageContainerBounds = value;
        RaisePropertyChanged("ImageContainerBounds");
    }
}

#endregion

#region View Model Instances

private ImageListViewModel ImageListVMInstance =
      (ImageListViewModel)Application.Current.FindResource("viewModel2");
private ImageProcessesViewModel ProcessesVMInstance =
      (ImageProcessesViewModel)Application.Current.FindResource("imageProcessesViewModel");

#endregion

#region StartKinectSensor() method

/// <summary>
/// Starts the Kinect sensor
/// </summary>
public void StartKinectSensor()
{
    if (ImageListVMInstance.ImageCollection == null || ImageListVMInstance.ImageCollection.Count == 0)
    {
        MessageBox.Show("Please load the images first before starting the sensor.");
        return;
    }

    try
    {
        KinectSensor.KinectSensors.StatusChanged += KinectSensors_StatusChanged;

        if (KinectSensor.KinectSensors.Count == 0)
        {
            KinectStatusColor = "FireBrick";
        }
        else
        {
            if (KinectSensor.KinectSensors[0].Status == KinectStatus.Connected)
            {
                // get the first Kinect sensor connected
                _sensor = KinectSensor.KinectSensors[0];

                // initialize the sensor
                Initialize();
            }
        }
    }
    catch (Exception e)
    {
        MessageBox.Show(e.ToString());
    }
}

#endregion

#region Initialize() method

/// <summary>
/// Initializes the Kinect sensor and its resources
/// </summary>
private void Initialize()
{
    if (_sensor == null)
        return;

    KinectStatusColor = "LimeGreen";
    UserStatusColor = "Gray";
    HideKinectIcon = true;

    _sensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
    _sensor.ColorFrameReady += _sensor_ColorFrameReady;

    _sensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
    _sensor.DepthFrameReady += _sensor_DepthFrameReady;

    _sensor.SkeletonStream.Enable(new TransformSmoothParameters
                                {
                                    Smoothing = 0.5f,
                                    Correction = 0.5f,
                                    Prediction = 0.5f,
                                    JitterRadius = 0.05f,
                                    MaxDeviationRadius = 0.04f
                                });
    _sensor.SkeletonFrameReady += _sensor_SkeletonFrameReady;

    _interactionStream = new InteractionStream(_sensor, new InteractionClient());
    _interactionStream.InteractionFrameReady += _interactionStream_InteractionFrameReady;

    _stabilityChecker = new StabilityChecker();
    _waveDetector = new WaveGestureDetector();

    _gestureDetector = new GestureDetector();
    _gestureDetector.OnGestureDetected += GestureDetector_OnGestureDetected;

    _sensor.Start();
}

#endregion

#region CleanUpResources() method

/// <summary>
/// Cleans up all the resources including the event handlers
/// </summary>
public void CleanUpResources()
{
    if (_gestureDetector != null)
        _gestureDetector.OnGestureDetected -= GestureDetector_OnGestureDetected;

    if (_interactionStream != null)
        _interactionStream.InteractionFrameReady -= _interactionStream_InteractionFrameReady;

    if (_sensor != null)
    {
        _sensor.ColorFrameReady -= _sensor_ColorFrameReady;
        _sensor.DepthFrameReady -= _sensor_DepthFrameReady;
        _sensor.SkeletonFrameReady -= _sensor_SkeletonFrameReady;

        _sensor.Stop();
        _sensor.Dispose();
        _sensor = null;
    }
}

#endregion

#region ResetImageProcesses() method

/// <summary>
/// Resets all image process to false
/// </summary>
private void ResetImageProcesses()
{
    ProcessesVMInstance.ZoomIn = false;
    ProcessesVMInstance.ZoomOut = false;
    ProcessesVMInstance.PanRight = false;
    ProcessesVMInstance.PanLeft = false;
    ProcessesVMInstance.PanUp = false;
    ProcessesVMInstance.PanDown = false;
    ProcessesVMInstance.IncreaseBrightness = false;
    ProcessesVMInstance.DecreaseBrightness = false;
    ProcessesVMInstance.RotateRight = false;
    ProcessesVMInstance.RotateLeft = false;
    ProcessesVMInstance.LimitAnimation = false;
}

#endregion

#region KinectSensors_StatusChanged()

/// <summary>
/// Event that is called when the Kinect status has been changed
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void KinectSensors_StatusChanged(object sender, StatusChangedEventArgs e)
{
    switch (e.Status)
    {
        case KinectStatus.Initializing:
            KinectStatusColor = "Gold";
            UserStatusColor = "Gray";
            break;

        case KinectStatus.Connected:
            if (_sensor == null)
            {
                _sensor = e.Sensor;
                Initialize();
            }
            break;

        case KinectStatus.Disconnected:
            if (_sensor == e.Sensor)
            {
                CleanUpResources();
                KinectStatusColor = "FireBrick";
                UserStatusColor = "Gray";
            }
            break;

        case KinectStatus.NotPowered:
            if (_sensor == e.Sensor)
            {
                CleanUpResources();
                KinectStatusColor = "FireBrick";
                UserStatusColor = "Gray";
                MessageBox.Show("The Kinect is plugged into the computer with its USB connection, but the
                      power plug appears to be not powered.");
            }
            break;

        case KinectStatus.InsufficientBandwidth:
            if (_sensor == e.Sensor)
            {
                CleanUpResources();
                KinectStatusColor = "FireBrick";
                UserStatusColor = "Gray";
                MessageBox.Show("There are too many USB devices plugged in. Please unplug one or more.");
            }
            break;

        default:
            MessageBox.Show("An unexpected error occurred. Please restart the program.");
            break;
    }
}

#endregion

#region _sensor_ColorFrameReady()

/// <summary>
/// Event that is called to stream color or video data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void _sensor_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
{
    if (_sensor == null)
```

```
            return;

        using (ColorImageFrame frame = e.OpenColorImageFrame())
        {
            if (frame == null)
                return;

            Update(frame);
        }
    }

    #endregion

    #region _sensor_DepthFrameReady()

    /// <summary>
    /// Event that is called to stream depth data
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    void _sensor_DepthFrameReady(object sender, DepthImageFrameReadyEventArgs e)
    {
        if (_sensor == null)
            return;

        using (DepthImageFrame frame = e.OpenDepthImageFrame())
        {
            if (frame == null)
                return;

            // pass to interaction stream
            _interactionStream.ProcessDepth(frame.GetRawPixelData(), frame.Timestamp);
        }
    }

    #endregion

    #region _sensor_SkeletonFrameReady()

    /// <summary>
    /// Event that is called to stream skeleton data
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    void _sensor_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
    {
        if (_sensor == null)
            return;

        using (SkeletonFrame frame = e.OpenSkeletonFrame())
        {
            if (frame == null)
                return;

            // copy frame data to list of skeletons
            skeletonList = new Skeleton[frame.SkeletonArrayLength];
            frame.CopySkeletonDataTo(skeletonList);

            // lock a single skeleton
            if (_currentTrackingId != 0)
            {
                _skeleton = (from trackskeleton in skeletonList
                             where trackskeleton.TrackingState == SkeletonTrackingState.Tracked
                             && trackskeleton.TrackingId == _currentTrackingId
                             select trackskeleton).FirstOrDefault();

                if (_skeleton == null)
                {
                    UserStatusColor = "Gray";

                    _currentTrackingId = 0;
                    _sensor.SkeletonStream.AppChoosesSkeletons = false;
                }
            }
            else
            {
                _skeleton2 = (from trackskeleton in skeletonList
                              where trackskeleton.TrackingState == SkeletonTrackingState.Tracked
                              select trackskeleton).FirstOrDefault();

                if (_skeleton2 != null) UserStatusColor = "Gold";
                else UserStatusColor = "Gray";

                _skeleton = (from trackskeleton in skeletonList
                             where trackskeleton.TrackingState == SkeletonTrackingState.Tracked
                             && (_waveDetector.WavePerformedRight(trackskeleton) ||
                                  _waveDetector.WavePerformedLeft(trackskeleton))
                             select trackskeleton).FirstOrDefault();

                if (_skeleton != null)
                {
                    _currentTrackingId = _skeleton.TrackingId;
                    _sensor.SkeletonStream.AppChoosesSkeletons = true;
                    _sensor.SkeletonStream.ChooseSkeletons(_currentTrackingId);
                }
            }

            Vector4 accelerometerReading = _sensor.AccelerometerGetCurrentReading();
            _interactionStream.ProcessSkeleton(skeletonList, accelerometerReading, frame.Timestamp);
            if (_skeleton == null || _skeleton.TrackingState == SkeletonTrackingState.NotTracked)
                return;

            DrawSkeleton(_skeleton);

            ResetImageProcesses();

            _stabilityChecker.AddPosition(_skeleton.TrackingId, Vector3D.ToVector3D(_skeleton.Position));

            if (!(_stabilityChecker.IsSkeletonStable(_skeleton.TrackingId) &&
                 _stabilityChecker.AreShouldersTowardSensor(_skeleton)) ||
                (_skeleton.Joints[JointType.HandRight].TrackingState != JointTrackingState.Tracked ||
                    _skeleton.Joints[JointType.HandLeft].TrackingState != JointTrackingState.Tracked) ||
                (_skeleton.Joints[JointType.HandRight].Position.Y <
                     _skeleton.Joints[JointType.HipRight].Position.Y &&
                    _skeleton.Joints[JointType.HandLeft].Position.Y <
                         _skeleton.Joints[JointType.HipLeft].Position.Y) ||
                (_skeleton.Joints[JointType.HandRight].Position.Y >=
                     _skeleton.Joints[JointType.HipRight].Position.Y &&
                    _skeleton.Joints[JointType.HandLeft].Position.Y >=
                         _skeleton.Joints[JointType.HipLeft].Position.Y))
            {
                UserStatusColor = "FireBrick";
                return;
            }

            UserStatusColor = "LimeGreen";

            _gestureDetector.AddGestureTrace(Vector3D.ToVector3D(_skeleton.Joints[JointType.HandRight].Position),
                 JointType.HandRight);
            _gestureDetector.AddGestureTrace(Vector3D.ToVector3D(_skeleton.Joints[JointType.HandLeft].Position),
                 JointType.HandLeft);
        }
    }

    #endregion

    #region _interactionStream_InteractionFrameReady()

    /// <summary>
    /// Event that is called to stream the interaction frames
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    void _interactionStream_InteractionFrameReady(object sender, InteractionFrameReadyEventArgs e)
    {
        if (_sensor == null)
            return;

        using (InteractionFrame frame = e.OpenInteractionFrame())
        {
            if (frame == null)
                return;

            userInfoList = new UserInfo[InteractionFrame.UserInfoArrayLength];
            frame.CopyInteractionDataTo(userInfoList);
        }

        if (_currentTrackingId == 0)
            return;

        UserInfo userInfo = (from user in userInfoList
                             where user.SkeletonTrackingId == _currentTrackingId
                             select user).FirstOrDefault();

        if (userInfo == null)
            return;

        foreach (InteractionHandPointer handPointer in userInfo.HandPointers)
        {
            if(handPointer.HandType == InteractionHandType.Left)
            {
                if (handPointer.HandEventType == InteractionHandEventType.Grip)
                {
                    leftHandClosed = true;
                }
                else if (handPointer.HandEventType == InteractionHandEventType.GripRelease)
                {
                    leftHandClosed = false;
                }
            }

            if (handPointer.HandType == InteractionHandType.Right)
            {
                if (handPointer.HandEventType == InteractionHandEventType.Grip)
                {
                    rightHandClosed = true;
                }
                else if (handPointer.HandEventType == InteractionHandEventType.GripRelease)
                {
                    rightHandClosed = false;
                }
            }
        }

        LeftHand = leftHandClosed;
        RightHand = rightHandClosed;
    }
```

```
#endregion

#region GestureDetector_OnGestureDetected()

/// <summary>
/// Event that is called every time a gesture is detected
/// </summary>
/// <param name="obj"></param>
void GestureDetector_OnGestureDetected(string obj)
{
    switch (obj)
    {
        case "SwipeUp":
            if (rightHandClosed)
            {
                if (ImageRenderBounds.Bottom <= ImageContainerBounds.Bottom)
                {
                    ProcessesVMInstance.LimitAnimation = true;
                    break;
                }

                ProcessesVMInstance.PanUp = true;
                ProcessesVMInstance.ActionStatus = "Pan Up";
            }
            else
            {
                if (_zoomCounter + 1 <= 5)
                {
                    ProcessesVMInstance.ZoomIn = true;
                    ProcessesVMInstance.ActionStatus = "Zoom In";
                    _zoomCounter++;
                }
                else ProcessesVMInstance.LimitAnimation = true;
            }
            break;
        case "SwipeDown":
            if (leftHandClosed)
            {
                if (ImageRenderBounds.Top >= ImageContainerBounds.Top)
                {
                    ProcessesVMInstance.LimitAnimation = true;
                    break;
                }

                ProcessesVMInstance.PanDown = true;
                ProcessesVMInstance.ActionStatus = "Pan Down";
            }
            else
            {
                if (_zoomCounter - 1 >= 0)
                {
                    ProcessesVMInstance.ZoomOut = true;
                    ProcessesVMInstance.ActionStatus = "Zoom Out";
                    _zoomCounter--;
                }
                else ProcessesVMInstance.LimitAnimation = true;
            }
            break;
        case "SwipeToRight":
            if (rightHandClosed)
            {
                if (ImageRenderBounds.Left >= ImageContainerBounds.Left)
                {
                    ProcessesVMInstance.LimitAnimation = true;
                    break;
                }

                ProcessesVMInstance.PanRight = true;
                ProcessesVMInstance.ActionStatus = "Pan Right";
            }
            break;
        case "SwipeToRight2":
            if(!rightHandClosed)
                ImageListVMInstance.IncrementSelectedIndex();
            break;
        case "SwipeToLeft":
            if (leftHandClosed)
            {
                if (ImageRenderBounds.Right <= ImageContainerBounds.Right)
                {
                    ProcessesVMInstance.LimitAnimation = true;
                    break;
                }

                ProcessesVMInstance.PanLeft = true;
                ProcessesVMInstance.ActionStatus = "Pan Left";
            }
            break;
        case "SwipeToLeft3":
            if (!leftHandClosed)
                ImageListVMInstance.DecrementSelectedIndex();
            break;
        case "DiagonalUp":
            if (!rightHandClosed)
            {
                ProcessesVMInstance.IncreaseBrightness = true;
                ProcessesVMInstance.ActionStatus = "Inc. Brightness";
            }
```

```
                    break;
                case "DiagonalDown":
                    if (!leftHandClosed)
                    {
                        ProcessesVMInstance.DecreaseBrightness = true;
                        ProcessesVMInstance.ActionStatus = "Dec. Brightness";
                    }
                    break;
                case "ArcRight":
                    if (!rightHandClosed)
                    {
                        ProcessesVMInstance.RotateRight = true;
                        ProcessesVMInstance.ActionStatus = "Rotate Right";
                    }
                    break;
                case "ArcLeft":
                    if (!leftHandClosed)
                    {
                        ProcessesVMInstance.RotateLeft = true;
                        ProcessesVMInstance.ActionStatus = "Rotate Left";
                    }
                    break;
            }
        }

        #endregion
    }
}
```

KinectViewModel.ColorStream.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows;

namespace ImageViewer2
{
    public partial class KinectViewModel
    {
        #region Members

        private WriteableBitmap _bitmap;

        #endregion

        #region Properties

        /// <summary>
        /// The bitmap that loads the video data
        /// </summary>
        public WriteableBitmap Bitmap
        {
            get { return _bitmap; }
            private set { _bitmap = value; }
        }

        #endregion

        #region Update() method

        /// <summary>
        /// Updates the content of the bitmap every time the frame changes
        /// </summary>
        /// <param name="frame"></param>
        private void Update(ColorImageFrame frame)
        {
            // create byte buffer with size equal to frame size
            byte[] pixelData = new byte[frame.PixelDataLength];

            // copy content of frame to byte buffer
            frame.CopyPixelDataTo(pixelData);

            // create WriteableBitmap on first Update call
            if (Bitmap == null)
            {
                // Bitmap must be BGR32 with 96 dots per inches on x and y
                Bitmap = new WriteableBitmap(frame.Width, frame.Height, 96, 96, PixelFormats.Bgr32, null);
            }

            // copy buffer to WriteableBitmap
            int stride = Bitmap.PixelWidth * Bitmap.Format.BitsPerPixel / 8;
            Int32Rect dirtyRect = new Int32Rect(0, 0, Bitmap.PixelWidth, Bitmap.PixelHeight);
            Bitmap.WritePixels(dirtyRect, pixelData, stride, 0);

            // notify UI
            RaisePropertyChanged(() => Bitmap);
        }

        #endregion
    }
}
```

KinectViewModel.SkeletonStream.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Windows.Media;
using System.Windows.Shapes;
using Microsoft.Kinect;
using System.Collections.ObjectModel;

namespace ImageViewer2
{
    public partial class KinectViewModel
    {
        #region Members

        private ObservableCollection<Vector2D> _joints;
        private ObservableCollection<Line> _bones;

        #endregion

        #region Properties

        /// <summary>
        /// The list containing the user's joints
        /// </summary>
        public ObservableCollection<Vector2D> Joints
        {
            get { return _joints; }
            private set { _joints = value; }
        }

        /// <summary>
        /// The list containing the user's bones
        /// </summary>
        public ObservableCollection<Line> Bones
        {
            get { return _bones; }
            private set { _bones = value; }
        }

        #endregion

        #region DrawSkeleton() method

        /// <summary>
        /// Draws the skeleton of the user on the bitmap
        /// </summary>
        /// <param name="skeleton"></param>
        private void DrawSkeleton(Skeleton skeleton)
        {
            Joints = new ObservableCollection<Vector2D>();
            Bones = new ObservableCollection<Line>();

            if (skeleton.TrackingState == SkeletonTrackingState.Tracked)
            {
                Vector2D headPt = ConvertToScreenSpace(skeleton.Joints[JointType.Head].Position);
                Vector2D shoulderCenterPt =
                      ConvertToScreenSpace(skeleton.Joints[JointType.ShoulderCenter].Position);
                Vector2D shoulderLeftPt = ConvertToScreenSpace(skeleton.Joints[JointType.ShoulderLeft].Position);
                Vector2D elbowLeftPt = ConvertToScreenSpace(skeleton.Joints[JointType.ElbowLeft].Position);
                Vector2D wristLeftPt = ConvertToScreenSpace(skeleton.Joints[JointType.WristLeft].Position);
                Vector2D handLeftPt = ConvertToScreenSpace(skeleton.Joints[JointType.HandLeft].Position);
                Vector2D shoulderRightPt = ConvertToScreenSpace(skeleton.Joints[JointType.ShoulderRight].Position);
                Vector2D elbowRightPt = ConvertToScreenSpace(skeleton.Joints[JointType.ElbowRight].Position);
                Vector2D wristRightPt = ConvertToScreenSpace(skeleton.Joints[JointType.WristRight].Position);
                Vector2D handRightPt = ConvertToScreenSpace(skeleton.Joints[JointType.HandRight].Position);
                Vector2D spinePt = ConvertToScreenSpace(skeleton.Joints[JointType.Spine].Position);
                Vector2D hipCenterPt = ConvertToScreenSpace(skeleton.Joints[JointType.HipCenter].Position);

                Joints.Add(headPt);
                Joints.Add(shoulderCenterPt);
                Joints.Add(shoulderLeftPt);
                Joints.Add(elbowLeftPt);
                Joints.Add(wristLeftPt);
                Joints.Add(handLeftPt);
                Joints.Add(shoulderRightPt);
                Joints.Add(elbowRightPt);
                Joints.Add(wristRightPt);
                Joints.Add(handRightPt);
                Joints.Add(spinePt);
                Joints.Add(hipCenterPt);

                Bones.Add(new Line(headPt, shoulderCenterPt));
                Bones.Add(new Line(shoulderCenterPt, shoulderLeftPt));
                Bones.Add(new Line(shoulderLeftPt, elbowLeftPt));
                Bones.Add(new Line(elbowLeftPt, wristLeftPt));
                Bones.Add(new Line(wristLeftPt, handLeftPt));
                Bones.Add(new Line(shoulderCenterPt, shoulderRightPt));
                Bones.Add(new Line(shoulderRightPt, elbowRightPt));
                Bones.Add(new Line(elbowRightPt, wristRightPt));
                Bones.Add(new Line(wristRightPt, handRightPt));
                Bones.Add(new Line(shoulderCenterPt, spinePt));
                Bones.Add(new Line(spinePt, hipCenterPt));
            }

            RaisePropertyChanged(() => Joints);
            RaisePropertyChanged(() => Bones);
        }

        #endregion

        #region ConverToScreenSpace() method

        /// <summary>
        /// Converts the position values of the joints to the screen space
        /// </summary>
        /// <param name="skeletonPt"></param>
        /// <returns></returns>
        private Vector2D ConvertToScreenSpace(SkeletonPoint skeletonPt)
        {
            float x = 0;
            float y = 0;

            if (_sensor.ColorStream.IsEnabled)
            {
                ColorImagePoint colorPt = new CoordinateMapper(_sensor).MapSkeletonPointToColorPoint(skeletonPt,
                    ColorImageFormat.RgbResolution640x480Fps30);
                x = colorPt.X;
                y = colorPt.Y;
            }

            // adjust to actual size of canvas
            // (640x480) adjusted to (280x200)
            // change the constants depending on
            // the rendered size of canvas
            return new Vector2D((x / 640) * 280, (y / 480) * 200);
        }

        #endregion
    }
}
```

Line.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public class Line
    {
        /// <summary>
        /// First endpoint of the line
        /// </summary>
        public Vector2D VectorOne { get; set; }

        /// <summary>
        /// Second endpoint of the line
        /// </summary>
        public Vector2D VectorTwo { get; set; }

        /// <summary>
        /// Creates a line given two endpoints
        /// </summary>
        /// <param name="v1"></param>
        /// <param name="v2"></param>
        public Line(Vector2D v1, Vector2D v2)
        {
            VectorOne = v1;
            VectorTwo = v2;
        }
    }
}
```

Vector2D.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public class Vector2D
    {
        /// <summary>
        /// The x-coordinate value of the 2D vector
        /// </summary>
        public float X { get; set; }

        /// <summary>
        /// The y-coordinate value of the 2D vector
        /// </summary>
        public float Y { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="x"></param>
        /// <param name="y"></param>
        public Vector2D(float x, float y)
        {
            X = x;
            Y = y;
        }

        /// <summary>
        /// The zero vector
        /// </summary>
        public static Vector2D Zero
        {
            get { return new Vector2D(0, 0); }
        }

        /// <summary>
        /// Calculates the magnitude of the vector
        /// </summary>
        public float Length
        {
            get { return (float)Math.Sqrt((X * X) + (Y * Y)); }
        }

        /// <summary>
        /// Addition operation for 2 vectors
        /// </summary>
        /// <param name="vector1"></param>
        /// <param name="vector2"></param>
        /// <returns></returns>
        public static Vector2D operator +(Vector2D vector1, Vector2D vector2)
        {
            return new Vector2D(vector1.X + vector2.X, vector1.Y + vector2.Y);
        }

        /// <summary>
        /// Subtraction operation for 2 vectors
        /// </summary>
        /// <param name="vector1"></param>
        /// <param name="vector2"></param>
        /// <returns></returns>
        public static Vector2D operator -(Vector2D vector1, Vector2D vector2)
        {
            return new Vector2D(vector1.X - vector2.X, vector1.Y - vector2.Y);
        }

        /// <summary>
        /// Scalar multiplication operation for 2 vectors
        /// </summary>
        /// <param name="multiplier"></param>
        /// <param name="vector"></param>
        /// <returns></returns>
        public static Vector2D operator *(float multiplier, Vector2D vector)
        {
            return new Vector2D(multiplier * vector.X, multiplier * vector.Y);
        }

        /// <summary>
        /// Scalar multiplication for 2 vectors (overload)
        /// </summary>
        /// <param name="vector"></param>
        /// <param name="multiplier"></param>
        /// <returns></returns>
        public static Vector2D operator *(Vector2D vector, float multiplier)
        {
            return multiplier * vector;
        }

        /// <summary>
        /// Division operation for 2 vectors
        /// </summary>
        /// <param name="vector"></param>
        /// <param name="divisor"></param>
        /// <returns></returns>
        public static Vector2D operator /(Vector2D vector, float divisor)
        {
            return new Vector2D(vector.X / divisor, vector.Y / divisor);
        }
    }
}
```

Vector3D.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect;

namespace ImageViewer2
{
    public class Vector3D
    {
        /// <summary>
        /// The x-coordinate value of the 3D vector
        /// </summary>
        public float X { get; set; }

        /// <summary>
        /// The y-coordinate value of the 3D vector
        /// </summary>
        public float Y { get; set; }
```

```
/// <summary>
/// The z-coordinate value of the 3D vector
/// </summary>
public float Z { get; set; }

/// <summary>
/// Constructor
/// </summary>
/// <param name="x"></param>
/// <param name="y"></param>
/// <param name="z"></param>
public Vector3D(float x, float y, float z)
{
    X = x;
    Y = y;
    Z = z;
}

/// <summary>
/// The zero vector
/// </summary>
public static Vector3D Zero
{
    get { return new Vector3D(0, 0, 0); }
}

/// <summary>
/// Calculates the magnitude of the vector
/// </summary>
public float Length
{
    get { return (float)Math.Sqrt((X * X) + (Y * Y) + (Z * Z)); }
}

/// <summary>
/// Addition operation for 2 vectors
/// </summary>
/// <param name="vector1"></param>
/// <param name="vector2"></param>
/// <returns></returns>
public static Vector3D operator +(Vector3D vector1, Vector3D vector2)
{
    return new Vector3D(vector1.X + vector2.X, vector1.Y + vector2.Y, vector1.Z + vector2.Z);
}

/// <summary>
/// Subtraction operation for 2 vectors
/// </summary>
/// <param name="vector1"></param>
/// <param name="vector2"></param>
/// <returns></returns>
public static Vector3D operator -(Vector3D vector1, Vector3D vector2)
{
    return new Vector3D(vector1.X - vector2.X, vector1.Y - vector2.Y, vector1.Z - vector2.Z);
}

/// <summary>
/// Multiplication operation for 2 vectors
/// </summary>
/// <param name="multiplier"></param>
/// <param name="vector"></param>
/// <returns></returns>
public static Vector3D operator *(float multiplier, Vector3D vector)
{
    return new Vector3D(multiplier * vector.X, multiplier * vector.Y, multiplier * vector.Z);
}

/// <summary>
/// Multiplication operation for 2 vectors (overload)
/// </summary>
/// <param name="vector"></param>
/// <param name="multiplier"></param>
/// <returns></returns>
public static Vector3D operator *(Vector3D vector, float multiplier)
{
    return multiplier * vector;
}

/// <summary>
/// Division operation for 2 vectors
/// </summary>
/// <param name="vector"></param>
/// <param name="divisor"></param>
/// <returns></returns>
public static Vector3D operator /(Vector3D vector, float divisor)
{
    return new Vector3D(vector.X / divisor, vector.Y / divisor, vector.Z / divisor);
}

/// <summary>
/// Converts a SkeletonPoint to a 3D vector
/// </summary>
/// <param name="point"></param>
/// <returns></returns>
public static Vector3D ToVector3D(SkeletonPoint point)
{
    return new Vector3D(point.X, point.Y, point.Z);
}
```

```
        /// <summary>
        /// Converts a 3D vector to a 2D vector by removing the z-coordinate value
        /// </summary>
        /// <returns></returns>
        public Vector2D ToVector2D()
        {
            return new Vector2D(this.X, this.Y);
        }
    }
}
```

InteractionClient.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect.Toolkit.Interaction;

namespace ImageViewer2
{
    public class InteractionClient : IInteractionClient
    {
        /// <summary>
        /// Gets the interaction points from the screen space
        /// </summary>
        /// <param name="skeletonTrackingId"></param>
        /// <param name="handType"></param>
        /// <param name="x"></param>
        /// <param name="y"></param>
        /// <returns></returns>
        public InteractionInfo GetInteractionInfoAtLocation(int skeletonTrackingId, InteractionHandType handType,
             double x, double y)
        {
            InteractionInfo result = new InteractionInfo();
            result.IsGripTarget = true;
            result.IsPressTarget = false;

            return result;
        }
    }
}
```

WaveGestureDetector.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect;

namespace ImageViewer2
{
    public class WaveGestureDetector
    {
        private static readonly int MinimumTime = 500;

        /// <summary>
        /// Contains the wave gesture segments of the right hand
        /// </summary>
        private bool[] WaveSegmentsRight { get; set; }

        /// <summary>
        /// Contains the wave gesture segment of the left hand
        /// </summary>
        private bool[] WaveSegmentsLeft { get; set; }

        /// <summary>
        /// The time the first wave gesture segment from the right hand occurred
        /// </summary>
        private DateTime TimeOfFirstSegment { get; set; }

        /// <summary>
        /// The time the second wave gesture segment from the right hand occurred
        /// </summary>
        private DateTime TimeOfSecondSegment { get; set; }

        /// <summary>
        /// The time the first wave gesture segment from the left hand occurred
        /// </summary>
        private DateTime TimeOfFirstSegment2 { get; set; }

        /// <summary>
        /// The time the second wave gesture segment from the left hand occurred
        /// </summary>
        private DateTime TimeOfSecondSegment2 { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        public WaveGestureDetector()
        {
            WaveSegmentsRight = new bool[] { false, false, false };
            WaveSegmentsLeft = new bool[] { false, false, false };
        }
```

```
/// <summary>
/// Resets the wave gesture segments from the right hand
/// </summary>
private void ResetSegmentsRight()
{
    for (int i = 0; i < WaveSegmentsRight.Length; i++)
        WaveSegmentsRight[i] = false;
}

/// <summary>
/// Resets the wave gesture segments from the left hand
/// </summary>
private void ResetSegmentsLeft()
{
    for (int i = 0; i < WaveSegmentsRight.Length; i++)
        WaveSegmentsLeft[i] = false;
}

/// <summary>
/// Generic method for checking whether a wave gesture has been performed
/// </summary>
/// <param name="user"></param>
/// <param name="handType"></param>
/// <param name="ConditionOne1"></param>
/// <param name="ConditionTwo1"></param>
/// <param name="ConditionTwo2"></param>
/// <param name="ConditionThree1"></param>
/// <param name="ConditionThree2"></param>
/// <param name="StartCondition"></param>
/// <param name="FinalCondition"></param>
/// <returns></returns>
private bool WavePerformed(Skeleton user, bool handType,
    Func<Skeleton, bool> ConditionOne1,
    Func<Skeleton, bool> ConditionTwo1, Func<bool> ConditionTwo2,
    Func<Skeleton, bool> ConditionThree1, Func<bool> ConditionThree2,
    Func<Skeleton, bool> StartCondition, Func<bool> FinalCondition)
{
    switch (handType)
    {
        case true:
            if (StartCondition(user))
            {
                if (ConditionOne1(user))
                {
                    TimeOfFirstSegment = DateTime.Now;
                    WaveSegmentsRight[0] = true;
                }
                else if (ConditionTwo1(user))
                {
                    if (ConditionTwo2())
                    {
                        TimeOfSecondSegment = DateTime.Now;
                        WaveSegmentsRight[1] = true;
                    }
                    else ResetSegmentsRight();
                }
                else if (ConditionThree1(user))
                {
                    if (ConditionThree2())
                        WaveSegmentsRight[2] = true;
                    else ResetSegmentsRight();
                }

                if (FinalCondition())
                {
                    ResetSegmentsRight();
                    return true;
                }
            }
            else ResetSegmentsRight();
            break;
        case false:
            if (StartCondition(user))
            {
                if (ConditionOne1(user))
                {
                    TimeOfFirstSegment2 = DateTime.Now;
                    WaveSegmentsLeft[0] = true;
                }
                else if (ConditionTwo1(user))
                {
                    if (ConditionTwo2())
                    {
                        TimeOfSecondSegment2 = DateTime.Now;
                        WaveSegmentsLeft[1] = true;
                    }
                    else ResetSegmentsLeft();
                }
                else if (ConditionThree1(user))
                {
                    if (ConditionThree2())
                        WaveSegmentsLeft[2] = true;
                    else ResetSegmentsLeft();
                }

                if (FinalCondition())
                {
                    ResetSegmentsLeft();
                    return true;
```

```
                        }
                    }
                    else ResetSegmentsLeft();
                    break;
            }

            return false;
        }

        /// <summary>
        /// Checks whether a wave gesture has been performed by the right hand
        /// </summary>
        /// <param name="user"></param>
        /// <returns></returns>
        public bool WavePerformedRight(Skeleton user)
        {
            if(WavePerformed(user, true,
                (u) => u.Joints[JointType.ElbowRight].Position.X < u.Joints[JointType.HandRight].Position.X
                         && !WaveSegmentsRight[0],
                (u) => u.Joints[JointType.ElbowRight].Position.X > u.Joints[JointType.HandRight].Position.X
                         && WaveSegmentsRight[0] && !WaveSegmentsRight[1],
                () => DateTime.Now.Subtract(TimeOfFirstSegment).Milliseconds < MinimumTime,
                (u) => u.Joints[JointType.ElbowRight].Position.X < u.Joints[JointType.HandRight].Position.X
                         && WaveSegmentsRight[0] && WaveSegmentsRight[1] && !WaveSegmentsRight[2],
                () => DateTime.Now.Subtract(TimeOfSecondSegment).Milliseconds < MinimumTime,
                (u) => u.Joints[JointType.ElbowRight].Position.Y < u.Joints[JointType.HandRight].Position.Y,
                () => WaveSegmentsRight[0] && WaveSegmentsRight[1] && WaveSegmentsRight[2]))
            {
                return true;
            }

            return false;
        }

        /// <summary>
        /// Checks whether a wave gesture has been performed by the left hand
        /// </summary>
        /// <param name="user"></param>
        /// <returns></returns>
        public bool WavePerformedLeft(Skeleton user)
        {
            if (WavePerformed(user, false,
                (u) => u.Joints[JointType.ElbowLeft].Position.X > u.Joints[JointType.HandLeft].Position.X
                       && !WaveSegmentsLeft[0],
                (u) => u.Joints[JointType.ElbowLeft].Position.X < u.Joints[JointType.HandLeft].Position.X
                       && WaveSegmentsLeft[0] && !WaveSegmentsLeft[1],
                () => DateTime.Now.Subtract(TimeOfFirstSegment2).Milliseconds < MinimumTime,
                (u) => u.Joints[JointType.ElbowLeft].Position.X > u.Joints[JointType.HandLeft].Position.X
                       && WaveSegmentsLeft[0] && WaveSegmentsLeft[1] && !WaveSegmentsLeft[2],
                () => DateTime.Now.Subtract(TimeOfSecondSegment2).Milliseconds < MinimumTime,
                (u) => u.Joints[JointType.ElbowLeft].Position.Y < u.Joints[JointType.HandLeft].Position.Y,
                () => WaveSegmentsLeft[0] && WaveSegmentsLeft[1] && WaveSegmentsLeft[2]))
            {
                return true;
            }

            return false;
        }
    }
}
```

StabilityChecker.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect;

namespace ImageViewer2
{
    public class StabilityChecker
    {
        #region Members

        private Dictionary<int, List<Vector3D>> _positions;
        private int _frameCountLimit;
        private float _threshold;

        #endregion

        #region Properties

        /// <summary>
        /// Contains the user's tracking ID and his/her position
        /// </summary>
        public Dictionary<int, List<Vector3D>> Positions
        {
            get { return _positions; }
            private set { _positions = value; }
        }

        /// <summary>
        /// The maximum number of frames needed for checking the user's stability
        /// </summary>
        public int FrameCountLimit
        {
            get { return _frameCountLimit; }
            private set { _frameCountLimit = value; }
        }

        /// <summary>
        /// Threshold value for stability
        /// </summary>
        public float Threshold
        {
            get { return _threshold; }
            private set { _threshold = value; }
        }

        #endregion

        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="frameCountLimit"></param>
        /// <param name="threshold"></param>
        public StabilityChecker(int frameCountLimit = 40, float threshold = 0.10F)
        {
            Positions = new Dictionary<int, List<Vector3D>>();
            FrameCountLimit = frameCountLimit;
            Threshold = threshold;
        }

        #endregion

        #region AddPosition()

        /// <summary>
        /// Adds or updates the user’s position
        /// </summary>
        /// <param name="trackingId"></param>
        /// <param name="position"></param>
        public void AddPosition(int trackingId, Vector3D position)
        {
            // if the trackingId is not present in the data dictionary,
            // add this trackingId
            if (!Positions.ContainsKey(trackingId))
                Positions.Add(trackingId, new List<Vector3D>());

            // add the new Vector3D position of this trackingId
            Positions[trackingId].Add(position);

            // remove the first inserted position of this
            // trackingId if the number of positions exceeds
            // the specified frameCountLimit
            if (Positions[trackingId].Count > FrameCountLimit)
                Positions[trackingId].RemoveAt(0);
        }

        #endregion

        #region IsSkeletonStable()

        /// <summary>
        /// Checks whether the user is stable or not
        /// </summary>
        /// <param name="trackingId"></param>
        /// <returns></returns>
        public bool IsSkeletonStable(int trackingId)
        {
            List<Vector3D> currPositions = Positions[trackingId];

            // if the number of positions is not yet
            // equal to the frameCountLimit, not stable
            if (currPositions.Count != FrameCountLimit)
                return false;

            // get last position
            Vector3D current = currPositions[currPositions.Count - 1];

            // get the average of all the previous positions
            Vector3D average = Vector3D.Zero;
            for (int i = 0; i < currPositions.Count - 1; i++)
            {
                average += currPositions[i];
            }

            average /= (currPositions.Count - 1);

            // if the length of change in position exceeds
            // the threshold, not stable
            if ((average - current).Length > Threshold)
                return false;

            return true;
        }

        #endregion

        #region AreShouldersTowardSensor()

        /// <summary>
        /// Checks whether the two shoulder points of the user are approximately equal on the z-axis
        /// </summary>
```

```
        /// <param name="skeleton"></param>
        /// <returns></returns>
        public bool AreShouldersTowardSensor(Skeleton skeleton)
        {
            Vector3D shoulderLeftPt = Vector3D.ToVector3D(skeleton.Joints[JointType.ShoulderLeft].Position);
            Vector3D shoulderRightPt = Vector3D.ToVector3D(skeleton.Joints[JointType.ShoulderRight].Position);

            float leftZDistance = shoulderLeftPt.Z;
            float rightZDistance = shoulderRightPt.Z;

            if (Math.Abs(leftZDistance - rightZDistance) > Threshold)
                return false;

            return true;
        }

        #endregion
    }
}
```

GestureEntry.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public class GestureEntry
    {
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="position"></param>
        /// <param name="dateTime"></param>
        public GestureEntry(Vector3D position, DateTime dateTime)
        {
            Position = position;
            EntryTime = dateTime;
        }

        private DateTime _entryTime;
        private Vector3D _position;

        /// <summary>
        /// The time the gesture point was obtained
        /// </summary>
        public DateTime EntryTime
        {
            get { return _entryTime; }
            set { _entryTime = value; }
        }

        /// <summary>
        /// The position of the gesture point on the screen space
        /// </summary>
        public Vector3D Position
        {
            get { return _position; }
            set { _position = value; }
        }
    }
}
```

GestureDetector.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.Kinect;

namespace ImageViewer2
{
    public partial class GestureDetector
    {
        #region Members

        private GestureClassifier _classifier;
        private List<GestureEntry> _rightHandEntries;
        private List<GestureEntry> _leftHandEntries;
        private int _minDelayBetweenGestures;
        private int _traceCountLimit;
        private DateTime _timeOfLastGesture;
        private float _threshold;
        private float _threshold2;
        private List<Vector2D> _gesturePoints;

        public event Action<string> OnGestureDetected;

        #endregion

        #region Properties

        /// <summary>
        /// The GestureClassifier instance for classifying the gesture points
        /// </summary>
        public GestureClassifier Classifier
        {
            get { return _classifier; }
            set { _classifier = value; }
        }

        /// <summary>
        /// The gesture points obtained from the hand entries
        /// </summary>
        public List<Vector2D> GesturePoints
        {
            get { return _gesturePoints; }
            set { _gesturePoints = value; }
        }

        /// <summary>
        /// Threshold value for a gesture
        /// </summary>
        public float Threshold
        {
            get { return _threshold; }
            set { _threshold = value; }
        }

        /// <summary>
        /// Threshold value for a gesture
        /// </summary>
        public float Threshold2
        {
            get { return _threshold2; }
            set { _threshold2 = value; }
        }

        /// <summary>
        /// The gesture points from the right hand obtained from the color and skeleton data
        /// </summary>
        public List<GestureEntry> RightHandEntries
        {
            get { return _rightHandEntries; }
            private set { _rightHandEntries = value; }
        }

        /// <summary>
        /// The gesture points from the left hand obtained from the color and skeleton data
        /// </summary>
        public List<GestureEntry> LeftHandEntries
        {
            get { return _leftHandEntries; }
            private set { _leftHandEntries = value; }
        }

        /// <summary>
        /// The minimum delay between the occurrence of two gestures
        /// </summary>
        public int MinDelayBetweenGestures
        {
            get { return _minDelayBetweenGestures; }
            private set { _minDelayBetweenGestures = value; }
        }

        /// <summary>
        /// The hand entries count limit
        /// </summary>
        public int TraceCountLimit
        {
            get { return _traceCountLimit; }
            private set { _traceCountLimit = value; }
        }

        /// <summary>
        /// The time (milliseconds) the last gesture occurred
        /// </summary>
        public DateTime TimeOfLastGesture
        {
            get { return _timeOfLastGesture; }
            private set { _timeOfLastGesture = value; }
        }

        #endregion

        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="traceCountLimit"></param>
        public GestureDetector(int traceCountLimit = 20)
        {
            Classifier = new GestureClassifier();
            GesturePoints = new List<Vector2D>();

            RightHandEntries = new List<GestureEntry>();
            LeftHandEntries = new List<GestureEntry>();

            TraceCountLimit = traceCountLimit;
            MinDelayBetweenGestures = 1000;
            TimeOfLastGesture = DateTime.Now;
            Threshold = 0.04F;
            Threshold2 = 0.02f;

            // Swipe Properties
            MinSwipeLength = 0.24F;
            MinSwipeLength2 = 0.397F;
            MinSwipeLength3 = 0.38F;
            MinSwipeLength4 = 0.058f;
            MinSwipeDuration = 250; // 0.25 secs
            MaxSwipeDuration = 2000; // 2 secs

            // Arc Properties
            MinArcLength = 0.30F;
            MinArcDuration = 250; // 0.25 secs
            MaxArcDuration = 2000; // 2 secs
        }

        #endregion

        #region AddGestureTrace() method

        /// <summary>
        /// Obtains the hand entries from the color and skeleton data
        /// </summary>
        /// <param name="position"></param>
        /// <param name="handJoint"></param>
        public void AddGestureTrace(Vector3D position, JointType handJoint)
        {
            GestureEntry newEntry = new GestureEntry(position, DateTime.Now);

            if (handJoint == JointType.HandRight)
                RightHandEntries.Add(newEntry);

            if (handJoint == JointType.HandLeft)
                LeftHandEntries.Add(newEntry);

            // Remove old traces
            if (RightHandEntries.Count > TraceCountLimit)
                RightHandEntries.RemoveAt(0);

            if (LeftHandEntries.Count > TraceCountLimit)
                LeftHandEntries.RemoveAt(0);

            CheckForValidGesture();
        }

        #endregion

        #region RaiseGestureDetected() method

        /// <summary>
        /// Raises the OnGestureDetected event
        /// </summary>
        /// <param name="gesture"></param>
        public void RaiseGestureDetected(string gesture)
        {
            // check if current gesture is NOT too close to the previous one
            if (DateTime.Now.Subtract(TimeOfLastGesture).TotalMilliseconds > MinDelayBetweenGestures)
            {
                if (OnGestureDetected != null)
                    OnGestureDetected(gesture);

                TimeOfLastGesture = DateTime.Now;
            }

            RightHandEntries.Clear();
            LeftHandEntries.Clear();
        }

        #endregion

        #region RaiseGestureDetected2() method

        /// <summary>
        /// Raises the OnGestureDetected event (without minimum delay)
        /// </summary>
        /// <param name="gesture"></param>
        public void RaiseGestureDetected2(string gesture)
        {
            if (OnGestureDetected != null)
                OnGestureDetected(gesture);

            TimeOfLastGesture = DateTime.Now;

            RightHandEntries.Clear();
            LeftHandEntries.Clear();
        }

        #endregion

        #region ScanTracePositions() method

        /// <summary>
        /// Generic method for checking gesture contraints
        /// </summary>
        /// <param name="handEntries"></param>
        /// <param name="IsLengthOK"></param>
        /// <param name="IsDirectionOK"></param>
        /// <param name="IsDepthOK"></param>
        /// <param name="minTime"></param>
        /// <param name="maxTime"></param>
        /// <returns></returns>
        private bool ScanTracePositions(List<GestureEntry> handEntries,
            Func<Vector3D, Vector3D, bool> IsLengthOK,
            Func<Vector3D, Vector3D, bool> IsDirectionOK,
            Func<Vector3D, Vector3D, bool> IsDepthOK,
            int minTime, int maxTime)
        {
            int start = 0;
            if (handEntries.Count != 0)
                GesturePoints.Add(handEntries[start].Position.ToVector2D());

            for (int i = 1; i < handEntries.Count - 1; i++)
            {
                if (handEntries.Count != 0)
                    GesturePoints.Add(handEntries[i].Position.ToVector2D());

                if (!IsDepthOK(handEntries[start].Position, handEntries[i].Position) ||
                    !IsDirectionOK(handEntries[i].Position, handEntries[i + 1].Position))
                {
                    start = i;
                    GesturePoints.Clear();

                    if (handEntries.Count != 0)
                        GesturePoints.Add(handEntries[start].Position.ToVector2D());
                }

                if (IsLengthOK(handEntries[i].Position, handEntries[start].Position))
                {
                    double totalMilliseconds =
                        (handEntries[i].EntryTime - handEntries[start].EntryTime).TotalMilliseconds;

                    if (totalMilliseconds >= minTime && totalMilliseconds <= maxTime)
                        return true;
                }
            }

            GesturePoints.Clear();
            return false;
        }

        #endregion

        #region NormalizePoints() method

        /// <summary>
        /// Applies normalization to the gesture points for classification
        /// </summary>
        /// <param name="pts"></param>
        /// <returns></returns>
        private List<Vector2D> NormalizePoints(List<Vector2D> pts)
        {
            List<Vector2D> points = new List<Vector2D>(pts.Count);

            points = Normalizer.ToFixedPoints(pts);
            points = Normalizer.ScaleToReference(points);
            points = Normalizer.CenterToOrigin(points);

            return points;
        }

        #endregion

        #region CheckForValidGesture() method

        /// <summary>
        /// Checks what type of gesture has been performed
        /// </summary>
        public void CheckForValidGesture()
        {
            // Swipe to right
            if (IsSwipeRight())
            {
                RaiseGestureDetected("SwipeToRight");
                return;
            }

            if (IsSwipeRight2())
            {
                RaiseGestureDetected2("SwipeToRight2");
                return;
            }

            // Swipe to left
            if (IsSwipeLeft())
            {
                RaiseGestureDetected("SwipeToLeft");
                return;
            }

            if (IsSwipeLeft3())
            {
                RaiseGestureDetected2("SwipeToLeft3");
                return;
            }

            // Diagonal Up
            if (IsDiagonalUp())
```

```
            {
                RaiseGestureDetected("DiagonalUp");
                return;
            }

            // Diagonal Down
            if (IsDiagonalDown())
            {
                RaiseGestureDetected("DiagonalDown");
                return;
            }

            // Swipe up
            if (IsSwipeUp())
            {
                RaiseGestureDetected("SwipeUp");
                return;
            }

            // Swipe down
            if (IsSwipeDown())
            {
                RaiseGestureDetected("SwipeDown");
                return;
            }

            // Right Arc
            if (IsRightArc())
            {
                RaiseGestureDetected("ArcRight");
                return;
            }

            // Left Arc
            if (IsLeftArc())
            {
                RaiseGestureDetected("ArcLeft");
                return;
            }
        }

        #endregion
    }
}
```

GestureDetector.Swipe.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public partial class GestureDetector
    {
        #region Members

        private float _minSwipeLength;
        private float _minSwipeLength2;
        private float _minSwipeLength3;
        private float _minSwipeLength4;
        private int _minSwipeDuration;
        private int _maxSwipeDuration;

        #endregion

        #region Properties

        /// <summary>
        /// The minimum length required for a swipe gesture
        /// </summary>
        public float MinSwipeLength
        {
            get { return _minSwipeLength; }
            set { _minSwipeLength = value; }
        }

        /// <summary>
        /// The minimum length required for a swipe gesture
        /// </summary>
        public float MinSwipeLength2
        {
            get { return _minSwipeLength2; }
            set { _minSwipeLength2 = value; }
        }

        /// <summary>
        /// The minimum length required for a swipe gesture
        /// </summary>
        public float MinSwipeLength3
        {
            get { return _minSwipeLength3; }
            set { _minSwipeLength3 = value; }
        }

        /// <summary>
        /// The minimum length required for a swipe gesture
        /// </summary>
```

```
public float MinSwipeLength4
{
    get { return _minSwipeLength4; }
    set { _minSwipeLength4 = value; }
}

/// <summary>
/// The minimum duration required for a swipe gesture
/// </summary>
public int MinSwipeDuration
{
    get { return _minSwipeDuration; }
    set { _minSwipeDuration = value; }
}

/// <summary>
/// The maximum duration required for a swipe gesture
/// </summary>
public int MaxSwipeDuration
{
    get { return _maxSwipeDuration; }
    set { _maxSwipeDuration = value; }
}

#endregion

#region IsSwipeRight() method

/// <summary>
/// Determines if a swipe to right gesture has been performed (pan right)
/// </summary>
/// <returns></returns>
private bool IsSwipeRight()
{
    if (ScanTracePositions(RightHandEntries,
        (p1, p2) => Math.Abs(p2.X - p1.X) > MinSwipeLength,
        (p1, p2) => p2.X - p1.X > 0,
        (p1, p2) => Math.Abs(p2.Z - p2.Z) < Threshold,
        100, 1000))
    {
        if (GesturePoints.Count != 0)
        {
            GesturePoints = NormalizePoints(GesturePoints);
            Classifier.Classify(GesturePoints);

            if (Classifier.HorizontalProbability == Classifier.MaxProbability() &&
                Classifier.HorizontalProbability >= 0.93)
                return true;
        }
    }

    return false;
}

#endregion

#region IsSwipeRight2() method

/// <summary>
/// Determines if a swipe to right gesture has been performed (browse right)
/// </summary>
/// <returns></returns>
private bool IsSwipeRight2()
{
    if(ScanTracePositions(RightHandEntries,
        (p1, p2) => Math.Abs(p2.X - p1.X) > MinSwipeLength4,
        (p1, p2) => p2.X - p1.X > 0,
        (p1, p2) => Math.Abs(p2.Y - p1.Y) < Threshold2 && Math.Abs(p2.Z - p1.Z) < Threshold,
        MinSwipeDuration, MaxSwipeDuration))
    {
        return true;
    }

    return false;
}

#endregion

#region IsSwipeLeft() method

/// <summary>
/// Determines if a swipe to left gesture has been performed (pan left)
/// </summary>
/// <returns></returns>
private bool IsSwipeLeft()
{
    if (ScanTracePositions(LeftHandEntries,
        (p1, p2) => Math.Abs(p2.X - p1.X) > MinSwipeLength,
        (p1, p2) => p2.X - p1.X < 0,
        (p1, p2) => Math.Abs(p2.Z - p2.Z) < Threshold,
        MinSwipeDuration, MaxSwipeDuration))
    {
        if (GesturePoints.Count != 0)
        {
            GesturePoints.Reverse();
            GesturePoints = NormalizePoints(GesturePoints);
            Classifier.Classify(GesturePoints);

            if (Classifier.HorizontalProbability == Classifier.MaxProbability() &&
                Classifier.HorizontalProbability >= 0.93)
                return true;
        }
    }

    return false;
}

#endregion

#region IsSwipeLeft3() method

/// <summary>
/// Determines if a swipe to left gesture has been performed (browse left)
/// </summary>
/// <returns></returns>
private bool IsSwipeLeft3()
{
    if (ScanTracePositions(LeftHandEntries,
        (p1, p2) => Math.Abs(p2.X - p1.X) > MinSwipeLength4,
        (p1, p2) => p2.X - p1.X < 0,
        (p1, p2) => Math.Abs(p2.Y - p1.Y) < Threshold2 && Math.Abs(p2.Z - p1.Z) < Threshold,
        MinSwipeDuration, MaxSwipeDuration))
    {
        return true;
    }

    return false;
}

#endregion

#region IsSwipeUp() method

/// <summary>
/// Determines if a swipe up gesture has been performed
/// </summary>
/// <returns></returns>
private bool IsSwipeUp()
{
    if (ScanTracePositions(RightHandEntries,
        (p1, p2) => Math.Abs(p2.Y - p1.Y) > MinSwipeLength2,
        (p1, p2) => p2.Y - p1.Y > 0,
        (p1, p2) => Math.Abs(p2.Z - p2.Z) < Threshold,
        MinSwipeDuration, MaxSwipeDuration))
    {
        if (GesturePoints.Count != 0)
        {
            GesturePoints = NormalizePoints(GesturePoints);
            Classifier.Classify(GesturePoints);

            if (Classifier.VerticalProbability == Classifier.MaxProbability() &&
                Classifier.VerticalProbability >= 0.9)
                return true;
        }
    }

    return false;
}

#endregion

#region IsSwipeDown() method

/// <summary>
/// Determines if a swipe down gesture has been performed
/// </summary>
/// <returns></returns>
private bool IsSwipeDown()
{
    if (ScanTracePositions(LeftHandEntries,
        (p1, p2) => Math.Abs(p2.Y - p1.Y) > MinSwipeLength2,
        (p1, p2) => p2.Y - p1.Y < 0,
        (p1, p2) => Math.Abs(p2.Z - p2.Z) < Threshold,
        MinSwipeDuration, MaxSwipeDuration))
    {
        if (GesturePoints.Count != 0)
        {
            GesturePoints.Reverse();
            GesturePoints = NormalizePoints(GesturePoints);
            Classifier.Classify(GesturePoints);

            if (Classifier.VerticalProbability == Classifier.MaxProbability() &&
                Classifier.VerticalProbability >= 0.9)
                return true;
        }
    }

    return false;
}

#endregion

#region IsDiagonalUp() method

/// <summary>
/// Determines if a diagonal up gesture has been performed
/// </summary>
/// <returns></returns>
```

```
        private bool IsDiagonalUp()
        {
            if (ScanTracePositions(RightHandEntries,
                (p1, p2) => (p2 - p1).Length > MinSwipeLength3,
                (p1, p2) => p2.X - p1.X > 0 && p2.Y - p1.Y > 0,
                (p1, p2) => Math.Abs(p2.Z - p2.Z) < Threshold,
                MinArcDuration, MaxArcDuration))
            {
                if (GesturePoints.Count != 0)
                {
                    GesturePoints = NormalizePoints(GesturePoints);
                    Classifier.Classify(GesturePoints);

                    if (Classifier.DiagonalProbability == Classifier.MaxProbability() &&
                        Classifier.DiagonalProbability >= 0.96)
                        return true;
                }
            }

            return false;
        }

        #endregion

        #region IsDiagonalDown() method

        /// <summary>
        /// Determines if a diagonal down gesture has been performed
        /// </summary>
        /// <returns></returns>
        private bool IsDiagonalDown()
        {
            if (ScanTracePositions(LeftHandEntries,
                (p1, p2) => (p2 - p1).Length > MinSwipeLength3,
                (p1, p2) => p2.X - p1.X < 0 && p2.Y - p1.Y < 0,
                (p1, p2) => Math.Abs(p2.Z - p2.Z) < Threshold,
                MinArcDuration, MaxArcDuration))
            {
                if (GesturePoints.Count != 0)
                {
                    GesturePoints.Reverse();
                    GesturePoints = NormalizePoints(GesturePoints);
                    Classifier.Classify(GesturePoints);

                    if (Classifier.DiagonalProbability == Classifier.MaxProbability() &&
                        Classifier.DiagonalProbability >= 0.96)
                        return true;
                }
            }

            return false;
        }

        #endregion
    }
}
```

GestureDetector.Arc.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public partial class GestureDetector
    {
        #region Members

        private float _minArcLength;
        private int _minArcDuration;
        private int _maxArcDuration;

        #endregion

        #region Properties

        /// <summary>
        /// The minimum length required for an arc gesture
        /// </summary>
        public float MinArcLength
        {
            get { return _minArcLength; }
            set { _minArcLength = value; }
        }

        /// <summary>
        /// The minimum duration required for an arc gesture
        /// </summary>
        public int MinArcDuration
        {
            get { return _minArcDuration; }
            set { _minArcDuration = value; }
        }

        /// <summary>
        /// The maximum duration required for an arc gesture
        /// </summary>
        public int MaxArcDuration
        {
            get { return _maxArcDuration; }
            set { _maxArcDuration = value; }
        }

        #endregion

        #region IsRightArc() method

        /// <summary>
        /// Determines if a right arc gesture has been performed
        /// </summary>
        /// <returns></returns>
        private bool IsRightArc()
        {
            if (ScanTracePositions(RightHandEntries,
                 (p1, p2) => (p2 - p1).Length > MinArcLength,
                 (p1, p2) => p2.X - p1.X > 0,
                 (p1, p2) => Math.Abs(p2.Z - p1.Z) < 0.10F,//Threshold,
                MinArcDuration, MaxArcDuration))
            {
                if (GesturePoints.Count != 0)
                {
                    GesturePoints = NormalizePoints(GesturePoints);
                    Classifier.Classify(GesturePoints);

                    if (Classifier.ArcProbability == Classifier.MaxProbability() &&
                        Classifier.ArcProbability >= 0.9)
                        return true;
                }
            }

            return false;
        }

        #endregion

        #region IsLeftArc() method

        /// <summary>
        /// Determines if a left arc gesture has been performed
        /// </summary>
        /// <returns></returns>
        private bool IsLeftArc()
        {
            if (ScanTracePositions(LeftHandEntries,
                 (p1, p2) => (p2 - p1).Length > MinArcLength,
                 (p1, p2) => p2.X - p1.X < 0,
                 (p1, p2) => Math.Abs(p2.Z - p1.Z) < 0.10F,//Threshold,
                MinArcDuration, MaxArcDuration))
            {
                if (GesturePoints.Count != 0)
                {
                    GesturePoints.Reverse();
                    GesturePoints = NormalizePoints(GesturePoints);
                    Classifier.Classify(GesturePoints);

                    if (Classifier.ArcProbability == Classifier.MaxProbability() &&
                        Classifier.ArcProbability >= 0.9)
                        return true;
                }
            }

            return false;
        }

        #endregion
    }
}
```

Normalizer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ImageViewer2
{
    public class Normalizer
    {
        /// <summary>
        /// Constructor
        /// </summary>
        public Normalizer() { }

        /// <summary>
        /// Gets the sum of all the distances between the gesture points
        /// </summary>
        /// <param name="pts"></param>
        /// <returns></returns>
        private static float DistanceSum(List<Vector2D> pts)
        {
            float length = 0;

            for (int i = 1; i < pts.Count; i++)
            {
                length += (pts[i - 1] - pts[i]).Length;
            }

            return length;
        }

        /// <summary>
        /// Modifies the gesture points to a defined number of points
        /// </summary>
        /// <param name="pts"></param>
        /// <param name="n"></param>
        /// <returns></returns>
        public static List<Vector2D> ToFixedPoints(List<Vector2D> pts, int n = 11)
        {
            List<Vector2D> src = new List<Vector2D>(pts);

            List<Vector2D> dest = new List<Vector2D>();
            dest.Add(src[0]);

            float avgLength = DistanceSum(pts) / (n - 1);
            float currDistance = 0;

            for (int i = 1; i < src.Count; i++)
            {
                Vector2D pt1 = src[i - 1];
                Vector2D pt2 = src[i];

                float distance = (pt2 - pt1).Length;

                if ((currDistance + distance) >= avgLength)
                {
                    Vector2D newPt = pt1 + ((avgLength - currDistance) / distance) * (pt2 - pt1);
                    dest.Add(newPt);
                    src.Insert(i, newPt);
                    currDistance = 0;
                }
                else
                    currDistance += distance;
            }

            if (dest.Count < n)
                dest.Add(src[src.Count - 1]);

            return dest;
        }

        /// <summary>
        /// Scales the gesture points to a 1 x 1 reference graduation
        /// </summary>
        /// <param name="pts"></param>
        /// <returns></returns>
        public static List<Vector2D> ScaleToReference(List<Vector2D> pts)
        {
            List<Vector2D> result = new List<Vector2D>(pts);

            float minX = pts.Min(p => p.X);
            float minY = pts.Min(p => p.Y);
            float maxX = pts.Max(p => p.X);
            float maxY = pts.Max(p => p.Y);

            float scale = Math.Max(maxX - minX, maxY - minY);

            for (int i = 0; i < result.Count; i++)
            {
                if (scale != 0)
                {
                    result[i].X /= scale;
                    result[i].Y /= scale;
                }
            }

            return result;
        }

        /// <summary>
        /// Centers the gesture points to origin (0, 0)
        /// </summary>
        /// <param name="pts"></param>
        /// <returns></returns>
        public static List<Vector2D> CenterToOrigin(List<Vector2D> pts)
        {
            List<Vector2D> result = new List<Vector2D>(pts);
            Vector2D center = pts[5];
            for (int i = 0; i < result.Count; i++)
            {
                result[i] -= center;
            }

            return result;
        }
    }
}
```

GestureClassifier.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```csharp
namespace ImageViewer2
{
    public class GestureClassifier
    {
        #region Probability Variables

        /// <summary>
        /// The probability value for the horizontal swipe template
        /// </summary>
        public double HorizontalProbability { get; private set; }

        /// <summary>
        /// The probability value for the vertical swipe template
        /// </summary>
        public double VerticalProbability { get; private set; }

        /// <summary>
        /// The probability value for the diagonal swipe template
        /// </summary>
        public double DiagonalProbability { get; private set; }

        /// <summary>
        /// The probability value for the arc template
        /// </summary>
        public double ArcProbability { get; private set; }

        #endregion

        #region Gesture Template Patterns

        /// <summary>
        /// The horizontal swipe pattern
        /// </summary>
        private static List<Vector2D> HorizontalPattern = new List<Vector2D>();

        /// <summary>
        /// The vertical swipe pattern
        /// </summary>
        private static List<Vector2D> VerticalPattern = new List<Vector2D>();

        /// <summary>
        /// The diagonal swipe pattern
        /// </summary>
        private static List<Vector2D> DiagonalPattern = new List<Vector2D>();

        /// <summary>
        /// The arc pattern
        /// </summary>
        private static List<Vector2D> ArcPattern = new List<Vector2D>();

        #endregion

        #region GenerateTemplates() method

        /// <summary>
        /// Generates the gesture templates by assigning values
        /// </summary>
        private void GenerateTemplates()
        {
            for (float i = -0.5f; i <= 0.5f; i = i + 0.1f)
            {
                HorizontalPattern.Add(new Vector2D(i, 0.0f));
                VerticalPattern.Add(new Vector2D(0.0f, i));
                DiagonalPattern.Add(new Vector2D(i, i));
                ArcPattern.Add(new Vector2D(i, -(i * i)));
            }
        }

        #endregion

        #region Constructor

        /// <summary>
        /// Constructor
        /// </summary>
        public GestureClassifier()
        {
            HorizontalProbability = 0.0;
            VerticalProbability = 0.0;
            DiagonalProbability = 0.0;
            ArcProbability = 0.0;

            GenerateTemplates();
        }

        #endregion

        #region Classify() method

        /// <summary>
        /// Computes for the probabilities for each gesture templates
        /// </summary>
        /// <param name="inputVector"></param>
        public void Classify(List<Vector2D> inputVector)
        {
            double sum;

            sum = 0;
            for (int i = 0; i < HorizontalPattern.Count; i++)
            {
                double x2 = Math.Pow(HorizontalPattern[i].X - inputVector[i].X, 2);
                double y2 = Math.Pow(HorizontalPattern[i].Y - inputVector[i].Y, 2);
                double exponent = (-(x2 + y2) / 2.0);
                sum += Math.Exp(exponent);

            }
            HorizontalProbability = (sum / HorizontalPattern.Count);

            sum = 0;
            for (int i = 0; i < VerticalPattern.Count; i++)
            {
                double x2 = Math.Pow(VerticalPattern[i].X - inputVector[i].X, 2);
                double y2 = Math.Pow(VerticalPattern[i].Y - inputVector[i].Y, 2);
                double exponent = (-(x2 + y2) / 2.0);
                sum += Math.Exp(exponent);

            }
            VerticalProbability = (sum / VerticalPattern.Count);

            sum = 0;
            for (int i = 0; i < DiagonalPattern.Count; i++)
            {
                double x2 = Math.Pow(DiagonalPattern[i].X - inputVector[i].X, 2);
                double y2 = Math.Pow(DiagonalPattern[i].Y - inputVector[i].Y, 2);
                double exponent = (-(x2 + y2) / 2.0);
                sum += Math.Exp(exponent);
            }
            DiagonalProbability = (sum / DiagonalPattern.Count);

            sum = 0;
            for (int i = 0; i < ArcPattern.Count; i++)
            {
                double x2 = Math.Pow(ArcPattern[i].X - inputVector[i].X, 2);
                double y2 = Math.Pow(ArcPattern[i].Y - inputVector[i].Y, 2);
                double exponent = (-(x2 + y2) / 2.0);
                sum += Math.Exp(exponent);
            }
            ArcProbability = (sum / ArcPattern.Count);
        }

        #endregion

        #region MaxProbability() method

        /// <summary>
        /// Gets the maximum probability
        /// </summary>
        /// <returns></returns>
        public double MaxProbability()
        {
            return Math.Max(HorizontalProbability, Math.Max(VerticalProbability, Math.Max(DiagonalProbability,
                   ArcProbability)));
        }

        #endregion
    }
}
```

BrightnessEffect.cs

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Effects;

namespace BrightnessEffect
{
    public class BrightnessEffect : ShaderEffect
    {
        #region Constructors

        static BrightnessEffect()
        {
            _pixelShader.UriSource = Global.MakePackUri("brightness.ps");
        }

        public BrightnessEffect()
        {
            this.PixelShader = _pixelShader;

            // Update each DependencyProperty that's registered with a shader register.  This
            // is needed to ensure the shader gets sent the proper default value.
            UpdateShaderValue(InputProperty);
            UpdateShaderValue(BrightnessProperty);
        }

        #endregion

        #region Dependency Properties

        public Brush Input
        {
            get { return (Brush)GetValue(InputProperty); }
            set { SetValue(InputProperty, value); }
        }

        // Brush-valued properties turn into sampler-property in the shader.
        // This helper sets "ImplicitInput" as the default, meaning the default
        // sampler is whatever the rendering of the element it's being applied to is.
        public static readonly DependencyProperty InputProperty =
            ShaderEffect.RegisterPixelShaderSamplerProperty("Input", typeof(BrightnessEffect), 0);

        public double Brightness
        {
            get { return ((double)(this.GetValue(BrightnessProperty))); }
            set { SetValue(BrightnessProperty, value); }
        }

        public static readonly DependencyProperty BrightnessProperty =
            DependencyProperty.Register("Brightness", typeof(double), typeof(BrightnessEffect), new
                 UIPropertyMetadata(((double)(0D)), PixelShaderConstantCallback(0)));

        #endregion

        #region Member Data

        private static PixelShader _pixelShader = new PixelShader();

        #endregion

    }
}
```

brightness.fx

```
sampler2D Texture1Sampler : register(S0);
float Brightness : register(C0);

float4 main(float2 uv : TEXCOORD) : COLOR
{
   float4 pixelColor = tex2D(Texture1Sampler, uv);

  // Apply brightness.
  pixelColor.rgb += Brightness;

 return pixelColor;
}
```

XI. Acknowledgement

The following people have provided me with enough help and support in order for me to finish this project. Without them, I may have gone into the wrong path and may not be able to achieve my goals.

1. Prof. Greg Baes - for being my adviser, for pointing out and correcting the wrongs in my work and for accepting may last minute SP proposal submission.
2. Prof. Geoff Solano - for introducing and teaching AI specifically neural networks.
3. Prof. Julieta Go - for answering my questions regarding thesis writing.
4. Prof. Juliet Nabos - for telling me what are the necessary details to include in my SP document.
5. Prof. Mithun Jacob - for giving me a free copy of his research entitled "Context-based hand gesture recognition for the operating room" which became one of my primary references.

I would also like to thank my family, my father Josefino, my mother Emerita, and my brothers Jesso Ron and Jan Ronel; my college best friends, Jerson, Kim, Patrick and Jayrell; high school best friends, Michelle, Jerome, Emil and Nigel and the rest of the people not mentioned here but have contributed a lot not only in the development of this project but also in the development of me as a person. In the future, I hope that I can be worthy enough to be of help to you in case you need one.