# Motion Signal Extraction Framework for the Microsoft Kinect Camera: Point Cloud Registration and its Application as a Motion Correction Metric in PET/CT

By

**Alexander C Whitehead**

**University of Hull**


Submitted for the MSc in

Advanced Computer Science

September 18

Word Count: 26,382

*I, Alexander C Whitehead, confirm that the work presented in this dissertation is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.*



*Alexander C Whitehead*

UNIVERSITY OF Hull

# Contents

# List of Illustrations

# List of Tables

# List of Abbreviations

**2D** Two Dimensional

**3D** Three Dimensional

**4D** Four Dimensional

**ASCII** American Standard Code for Information Interchange

**CAD** Computer Aided Design

**CMOS** Complementary Metal Oxide Semiconductor

**CT** Computed Tomography

**FOV** Field Of View

**GNOME** GNU network object model environment

**GNU** GNU's Not Unix

**GPIO** General Purpose Input Output

**GUI** Graphical User Interface

**ICP** Iterative Closest Point

**ISO** International Organization For Standardization

**KPCLP** Kinect Point Cloud Library Processing

**MLEM** Maximum Likelihood Expectation Maximisation

**MR** Magnetic Resonance

**MRI** Magnetic Resonance Imaging

**NAN** Not A Number

**NDT** Normal Distributions Transform

**OSEM** Ordered Subset Expectation Maximisation

**PCD** Point Cloud Data

**PCL** Point Cloud Library

**PET** Positron Emission Tomography

**POSIX** Portable Operating System Interface

**RANSAC** Random Sample Consensus

**ROI** Region Of Interest

**ROIs** Regions Of Interest

**RGB** Red Green Blue

**RMSD** Root Mean Square Deviation

**RMSE** Root Mean Squared Error

**SAvVy** Sinogram Acquisition Viewer

**SOR** Statistical Outlier Removal

**SPECT** Single Photon Emission Computed Tomography

**STIR** Software For Tomographic Image Reconstruction

**TOF** Time Of Flight

**UML** Unified Modelling Language

**VFH** Viewpoint Feature Histogram

**VGF** Voxel Grid Filter

**XML** Extensible Markup Language

# 1 Introduction

Positron emission tomography (PET) is a common medical imaging modality used for acquiring functional images, images of the internal metabolic processes of a subject. These scans can take upwards of a few minutes to complete and during this time the subject of the scan could move for any number of reasons, including respiratory movement and musculoskeletal movement. Current standard practise is to attempt to ignore the movement of the subject, however, any movement no matter how slight degrades image resolution and introduces motion related artefacts.

In order to attempt to correct for the movement of the subject a method to track this motion must be found. This dissertation proposes a method to correct for subject movement in PET/CT scans, the method proposed involves using a stereo depth sensing camera to track the motion of the subject. The stereo depth sensing camera tracks the motion by taking depth images of the subject, in a scene, multiple times per second. The movement of the subject is then found by calculating the difference between these depth images.

Specifically, in order to track the change over time of the depth images, the application proposed in this dissertation will acquire data from the three dimensional stereo depth sensor of a Microsoft Kinect camera, the application will then use this to create a point cloud representation of a given scene. After the point cloud representation of the scene has been found registration will then be used to find the change in position of each point in each point cloud from point cloud to point cloud iteratively. The output of the registration processes will be used to calculate and output a translation or vector field that represents the change in the position of the objects in the scene over time.

Currently a stand alone, cross platform framework is under development to provide a facility for image analysis, reconstruction, the construction of an image of an object from multiple projections of the object from many different angles and data processing of a multitude of data formats, this framework is called the software for tomographic image reconstruction (STIR) framework. (Thielemans *et al.*, 2012) STIR will provide these facilities for data acquired through the use of PET, computed tomography (CT), single photon emission computed tomography (SPECT) and magnetic resonance (MR) or magnetic resonance imaging (MRI) scanners.

The development of this project will take place using and be compatible with the STIR image reconstruction toolkit version three and the sinogram acquisition viewer or STIR data viewer (SAvVy). (*NikEfth/SAvVy: SAvVy - Stir dAta Viewer*, 2018)

The goal of this project will be the development of modules or libraries that will provide the function of motion correction and can be integrated into STIR or SAvVy. These modules or libraries will provide the motion correction by attempting to extract translations or vector fields from, the change over time of, depth images. These depth images will be acquired using a Microsoft Kinect camera. The output from the application will then hopefully be used to warp the data of the medical scanners in order to correct for observed motion, this is known as motion correction or motion compensated reconstruction.

The original research proposal for this project can be seen in the appendix below. (Appendix A:)

A report detailing the ethical concerns of this project can be seen in the appendix below. (Appendix B:)

A table showing a breakdown of some risks which could occur while working on the project and then expanding upon these risks by determining the current risk, determining solutions to these risks and then determining the residual risk can be seen in the appendix below. (Appendix C:)

# 2 Aims, Hypothesis and Objectives

The aim of this project is:

*To develop a method of motion correction for medical imaging scanners, using the Microsoft Kinect camera, which can then be integrated into STIR or SAvVy*

The hypothesis for this project is that it is possible to extrapolate from a number of point clouds produced by a stereo three dimensional (3D) camera a metric that represents the motion of an object within a medical imaging scanner. It is then possible to use this metric to minimize or remove motion related artefacts from and improve the resolving ability of scan data produced using said medical imaging scanner.

*To achieve the aim and validate the hypothesis set out above the following objectives will be completed:*

1. *Read data from the Kinect camera.*
2. *Create a stand alone application to interface with the Kinect camera.*
3. *Add the ability to control the tilt motor of the Kinect camera.*
4. *Add the ability to extract depth and/or RGB images from the Kinect camera.*
5. *Add the ability to save depth and/or RGB images from the Kinect camera to text or binary file.*
6. *Add the ability to output multiple depth and/or RGB images per execution of the application.*
7. *Add the ability to timestamp output with timestamp relative to the Kinect camera.*
8. *Add the ability to change output settings for the depth and/or RGB images.*
9. *Add the ability to output header files for the depth and/or RGB images.*
10. *Create a stand alone application to load the header output from the Kinect camera application.*
11. *Add the ability to load data from header file.*
12. *Add the ability to calculate point clouds from the loaded data.*
13. *Add the ability to calculate centre of gravity for the point clouds.*

14. *Add the ability to visualise the point clouds and centres of gravity.*
15. *Add the ability to globally threshold the point clouds.*
16. *Add the ability to de noise the point clouds.*
17. *Add the ability to downsample the point clouds.*
18. *Add the ability to register between the point clouds.*
19. *Add the ability to use the output of previous registration step as an input to the next registration step.*
20. *Add the ability to extract motion signal from the registration output.*
21. *Add the ability to asses the amount of movement between the point clouds.*
22. *Add the ability to skip processing based on the amount of movement between the point clouds.*
23. *Add the ability to extract motion signal from the movement between the point clouds.*
24. *Add the ability to track a region of the point clouds.*
25. *Add the ability to extract the motion signal from the tracked region of the point clouds.*
26. *Add the ability to change the output settings and values for the registration step and the motion signal extraction.*
27. *Add the ability to save the values and output of the registration step and the motion signal extraction to a text or binary file.*

### Objective 1: Read data from the Kinect camera

An application should be created which can read data from the Kinect camera in real time and visualise this data to the user.

The open source OpenKinect Libfreenect or OpenKinect Libfreenect2 Kinect driver should be used for this application, this is because it allows for the development on and distribution to multiple different operating systems, opening up the eventual developer and user base for the application. (*FAQ - OpenKinect*, no date)

### Objective 2: Create a stand alone application to interface with the Kinect camera

An application should be created which can either be launched directly as an executable program or can be compiled as a library and then accessed within another application.

To achieve this the application could be written as a library and then a wrapper application could be written to run it independently.

This is advantageous as it allows for more rapid prototyping of ideas without having to rely on a large codebase, which in itself could contain errors, while also not hindering the ease with which the application could be integrated into a larger framework.

This application should be able to access the callbacks from the Kinect camera.

### Objective 3: Add the ability to control the tilt motor of the Kinect camera

The ability to send signals to the Kinect camera in order to request that it adjusts its tilt angle should be added, the ability to request the current tilt angle should also be considered.

An element should be added to the GUI to allow for the user to request that the Kinect camera's tilt is adjusted up or down by a preset amount, the ability to input an exact angle for the Kinect to tilt to should also be considered.

The Kinect camera's tilt angle should never be allowed to exceed safe parameters. Safe parameters meaning that the servo motor, used to tilt the Kinect camera, should not be pushed beyond the angle that it was manufactured to turn to or to the point that the body of the Kinect camera collides with the base of the camera.

This feature would be useful as it allows for the user to better define the area that the Kinect camera will be scanning.

### Objective 4: Add the ability to extract depth and/or RGB images from the Kinect camera

The ability for the application to read depth and/or red green blue (RGB) data which has been written into a buffer by the callbacks from the Kinect camera should be added.

Initially the callbacks from the Kinect camera can operate on the main thread, however it would be ideal to access the callbacks from their own thread as this would not only speed up the application and allow for fewer dropped frames from the Kinect camera but would also stop the callbacks from blocking input to the GUI.

The depth and/or RGB data should be held in memory in order to allow for further processing.

### Objective 5: Add the ability to save depth and/or RGB images from the Kinect camera to text or binary file

The ability to save the data from one scan of the Kinect camera should be added to the application.

The data should be saved to a file in storage as either a text file or a binary file.

An object should be created to represent the data output from the Kinect camera, this object should overload the streaming operators which will allow it to be written either automatically to the GUI or to a file.

### Objective 6: Add the ability to output multiple depth and/or RGB images per execution of the application

The ability to save the data of multiple concurrent scans from the Kinect camera should be added to the application.

It is possible to either save each piece of data as they are read from the Kinect camera or to save all pieces of data at once after the scan has been complete.

Each new piece of data should be saved into a list or vector contained within the object created to represent the data output from the Kinect camera, this will allow the data to be accessed again at a later point in the program.

After execution has ended, a method could be called which would stream the entire list or vector either to the console output or to a file.

### Objective 7: Add the ability to timestamp output with timestamp relative to the Kinect camera.

The ability to save the time that a scan occurred should be added to the application.

Each timestamp should be saved within the object created to represent the data output from the Kinect camera, this will allow all data relevant to a given scan to be contained together.

Timestamping the data from the Kinect camera is necessary to synchronize the Kinect camera output to the PET/CT scanner output later.

The timestamp saved should not only be relative to the Kinect camera but also relative to the time since epoch in order to be able to better identify data from different scans.

### Objective 8: Add the ability to change output settings for the depth and/or RGB images

The ability to change the output directories for the depth and/or RGB images should be added.

The ability to block the operation of either depth and/or RGB data callbacks should be offered to allow for the user to optimise the operation of the application for their own needs.

### Objective 9: Add the ability to output header files for the depth and/or RGB images

The ability to generate header files for the depth and/or RGB images should be added, these header files can be used later when reconstructing point clouds from the depth data.

Each header file should contain at least the file path to the image file, information regarding the resolution of the image, the bit depth of the image, the time at which the image was acquired and how many dimensions there are to the data.

### Objective 10: Create a stand alone application to load the header output from the Kinect camera application

An application should be created which can either be launched directly as an executable program or can be compiled as a library and then accessed within another application.

To achieve this the application could be written as a library and then a wrapper application could be written to run it independently.

This is advantageous as it allows for more rapid prototyping of ideas without having to rely on a large codebase, which in itself could contain errors, while also not hindering the ease with which the application could be integrated into a larger framework.

This application should be able to open the header files created by the Kinect camera application and load the information contained into an object representing the header file.

### Objective 11: Add the ability to load data from header file

The ability to load the depth and/or RGB data located at the file path specified in the header file should be added, this depth and/or RGB data should be loaded into a buffer which is located in the object that was created to represent the header file.

This data should be loaded at a point after the loading of the header file itself, this is because it can take a substantial amount of time to load the data from storage and as such the user should be certain that the correct header files have been selected before the data is extracted.

The size of the buffer to contain the data can be determined from the resolution and dimension information in the header file, the order that the data buffers should be loaded and stored can be determined from the timestamp located within the header file.

### Objective 12: Add the ability to calculate point clouds from the loaded data

The ability to calculate an unordered Cartesian point cloud from a depth image should be added.

Because perspective forces points to be displaced by greater amounts in the x and y axis as they are displaced in the z axis the algorithm to calculate a point cloud should take perspective into account when determining a point's position, this calculation should use the focal distance of the camera in order to estimate the displacement caused by the camera. This approach is far more accurate than just displacing each pixel by its depth in the z axis. (*Imaging Information - OpenKinect*, no date)

If the Kinect camera cannot detect a valid depth for a given pixel in the depth image this should be represented by a not a number (NAN) value in the point cloud, this is because it will be possible then to later iterate through the point cloud and remove any NAN values. This will not only reduce the amount of memory that is used by the application but will also speed up any subsequent calculations.

### Objective 13: Add the ability to calculate centre of gravity for the point clouds

The ability to calculate the centre of gravity of a point cloud as a centroid should be added.

This centre of gravity can be used later to aid in more naive calculations such as where the movement of the point cloud as a whole needs to be taken into account.

The centre of gravity of the point cloud can also be used as part of a visualisation to aid in the users understanding of the data that they have acquired. The centre of gravity could also be used to aid in the users assessment of how changes to the settings and parameters of the program affect the programs output.

### Objective 14: Add the ability to visualise the point clouds and centres of gravity

The ability to visualise the point cloud pairs post registration and the centre of gravity of these point clouds should be added.

This visualisation can be used to aid in the users understanding of the data that they have acquired, the visualisation could also be used to aid in the users assessment of how changes to the settings and parameters of the program affect the programs output.

The visualisation should be displayed for each registered pair of point clouds and should block the programs execution while running. The visualisation should be optional so as to allow the application to be run without user input.

This method should be extendable so that any number of point clouds and centres of gravity can be added at a later date.

The colours of the point clouds and centres of gravity should be able to be set externally so as to ensure that each point cloud and centre of gravity is distinguishable from each other, the point size of the point clouds and centres of gravity could also be set externally so as to tailor the visualisation to each users needs.

### Objective 15: Add the ability to globally threshold the point clouds

The ability to remove structures that are not regions of interest (ROIs) should be added to the application.

In every set of data from the Kinect camera it is likely there will be structures that are of no importance to the functionality of the application, for instance the bezel of the PET/CT scanner. These undesirable objects should be removed to cut down on processing time and memory usage. A smaller data set is also easier to visualise clearly.

There are multiple ways to achieve this result, a more naive method would be to define the region of interest (ROI) as being at a certain distance from the camera, any points that are beyond this threshold should be removed. To enhance this method a near and a far limit could be introduced, thus not only could points beyond a certain threshold be removed but points closer than another threshold could also be removed. This would make the ROI a thin bar.

A more complex solution would be to acquire a point cloud before scanning commences that only contains the objects that are to be removed from the data, therefore by subtracting the initial point cloud from every subsequent point cloud the bezel of the PET/CT scanner and other extraneous undesirable data or ROIs will be removed.

The parameters of the threshold or segmentation algorithm should be set externally so as to allow the user to optimise the threshold or segmentation algorithm to their own needs.

### Objective 16: Add the ability to de noise the point clouds

The ability to remove extraneous or outlying points which represent added noise from the data should be added to the application.

This could be achieved by iterating through every point in a given point cloud and finding for a selection of points around said initial point the standard deviation of this group of points, if the initial point is outside a given standard deviation of this group of points it should be considered to be noise and should be removed from the point cloud as a whole.

The point cloud library (PCL) contains methods and algorithms that have been implemented and tested to clean or de noise point clouds.

The parameters of the de noising algorithm, including the size of the group of points and the number of standard deviations that a given point can be from the group of points, should be set externally so as to allow the user to optimise the intensity and speed of the de noising algorithm to their own needs.

### Objective 17: Add the ability to downsample the point clouds

The ability to downsample point clouds should be added to the application.

The raw point cloud is likely to contain many more points than are necessary for the registration algorithm to accurately register point cloud to point cloud. For each unnecessary point in each given point cloud the time of execution of the application increases dramatically, therefore it is advantageous to downsample each point cloud.

One method to downsample a point cloud would be to pass a filter, of a given voxel size, over a given point cloud and then every point inside the voxel is averaged to one individual point.

The size of the voxel used to downsample each point cloud should be set externally so as to allow the user to optimise the intensity of the downsampling algorithm to their own needs.

### Objective 18: Add the ability to register between the point clouds

The ability to find the translation from one set of points to another should be added to the application, this is known as registration.

Registration is the process of finding a translation which moves all of the points in one data set to be as close as possible to the positions of points in another data set. This process optimises to produce as little error as possible, error is usually measured as mean square error.

The registration process can be either rigid or non rigid; this means that either the relationship between the points must remain the same for rigid registration or that the relationship between the points can change for non rigid registration. Generally, rigid registration will be applied to a pair of point clouds before a non rigid transformation is calculated in order to increase the accuracy of the non rigid registration step.

The PCL contains methods and algorithms that have been implemented and tested to register point clouds.

The parameters of the registration algorithm should be able to be set externally so as to allow the user to optimise the registration algorithm to their own needs, also the exact registration algorithm used should be selectable.

### Objective 19: Add the ability to use the output of previous registration step as an input to the next registration step

The ability to use the output of the previous registration step as the guess matrix for the next registration step should be added to the application.

This means that each subsequent registration step should start from the transformation matrix that was determined in the previous registration step. This is advantageous as ideally this application will predominantly be used to track objects and as such the objects will only move by a small amount from frame to frame, thus by starting each registration step from the end of the previous registration step fewer iterations of the registration algorithm should be required, subsequently the entire registration algorithm should compute faster and more accurately.

The parameters of the guess matrix should be able to be set externally so as to allow the user to optimise the registration algorithm to their own needs, also the ability to use the output of one registration step as the input to the next should be selectable.

### Objective 20: Add the ability to extract motion signal from the registration output

The ability to extract the transformation matrix or vector field from the registration algorithm should be added to the application.

The transformation matrix which is extracted from the rigid registration algorithm represents the main variable to be used in the motion correction of the PET/CT data, this matrix represents any rotation, scale, skew or translation that has occurred between the source and target point cloud as a homogeneous matrix.

If a non rigid registration algorithm is used a vector field with a transformation matrix for each individual point should be output instead.

This transformation matrix or vector field should be stored in some kind of buffer in order to be used in other parts of the application.

### Objective 21: Add the ability to asses the amount of movement between the point clouds

The ability to extract the distance between the centre of gravity of both point clouds or the ability to extract the amount of movement from both point clouds should be added to the application.

There are two main methods which could be used to calculate the amount of movement between two point clouds, a naive approach would be to determine the Euclidean linear distance between the centre of gravity of both point clouds, this could be calculated using the distance formula derived from the Pythagorean theorem. A more complex approach would be to use Eigenvectors.

The algorithm used to estimate the amount of movement between the two point clouds should be selected externally.

### Objective 22: Add the ability to skip processing based on the amount of movement between the point clouds

The ability to skip the registration step of the program based on a threshold on the amount of linear movement from point cloud to point cloud should be added to the application.

This is advantageous as a great deal of the overall processing time of the application will be dedicated to the registration step. Therefore, where the linear movement

between a point cloud pair is determined to fall below an acceptable threshold the registration step could be skipped in order to speed up the application. This would be especially useful where the data set which is to be registered is exceptionally large.

The parameters of the movement threshold should be able to be set externally so as to allow the user to optimise the amount of movement which will skip the registration steps execution to their own needs.

### Objective 23: Add the ability to extract motion signal from the movement between the point clouds

The ability to extract a motion signal from the difference between the centre of gravities of the two point clouds should be added.

The difference between the positions of the centre of gravities of two point clouds, represented as a vector, may very naively estimate a kind of surrogate breathing signal for the entire subject.

The vector containing the difference between the centre of gravities of the two point clouds should be added to a buffer in order to be used in other parts of the application.

### Objective 24: Add the ability to track a region of the point clouds

The ability to track a specific region of a point cloud should be added to the application.

A naive way to track a region of a point cloud would be, firstly to define a point at some distance away from the centre of gravity of the point cloud, then threshold the point cloud where every point in the point cloud where the Euclidean linear distance to the point which was defined in the previous step is below the threshold is added to the ROI.

A more complex method to track a region of a point cloud would be to define some surface in the first point cloud, then use feature matching to attempt to locate this surface in every subsequent point cloud.

The parameters of the tracking algorithm should be able to be set externally so as to allow the user to optimise the exact region which should be tracked to their own needs.

### Objective 25: Add the ability to extract the motion signal from the tracked region of the point clouds

The ability to extract a motion signal from the difference between the tracked regions of the point clouds should be added.

There are two main methods which could be used to extract the motion signal from the tracked region of the point clouds, a naive approach would be to find the difference between the positions of the centre of gravities of the tracked regions of the point clouds as a vector, this may estimate the surrogate breathing signal of a region of the subject.

A more complex approach would be to use registration in order to attempt to find the same pattern of points in the target point cloud as in the tracked region, this method has the advantage of returning a homogeneous transformation.

The vector or homogeneous transformation containing the difference between the tracked regions of the point clouds should be added to a buffer in order to be used in other parts of the application.

### Objective 26: Add the ability to change the output settings and values for the registration step and the motion signal extraction

The ability to change the parameters of the application should be added to the GUI of the application.

The parameters of the application should be able to be set externally in the GUI of the application so as to allow the user to optimise the application to their own needs.

### Objective 27: Add the ability to save the values and output of the registration step and the motion signal extraction to a text or binary file

The ability to save the output from the entire registration application should be added to the application.

The data should be saved either to a file in storage as a text file or a binary file.

An object should be created that can contain the output string of text from the registration application, this object should overload the streaming operators allowing it to be written either automatically to the GUI or to a file.

The ability to change the output directories for the output should be added.

The ability to block the output of either text or binary output should be offered to allow for the user to optimise the operation of the application for their own needs.

A header should be added to the output from the registration application, this header should contain the parameters which were used in that particular set of calculations, this should allow experiments to be repeated and conclusions to be more accurately drawn.

The task list and time plan which have been determined from the above aims, hypothesis and objectives can be seen in the appendix below. (Appendix D:) (Appendix E:) (Appendix F:)

The initial aims, hypothesis and objectives can be seen in the appendix below. (Appendix G:)

The initial task list and time plan which were determined from the initial aims, hypothesis and objectives can be seen in the appendix below. (Appendix H:) (Appendix I:) (Appendix J:)

# 3 Background and Methods

## 3.1 Problem Context

### Positron Emission Tomography: Overview

PET is a medical imaging modality that is used to acquire functional images, relating to the internal metabolic processes of a subjects body. These images are acquired as an aid to the diagnosis of irregularities within and the study of the body. These images are commonly used in the fields of Oncology, Cardiology, Neurology and Psychiatry. (Bertolli, 2018)

In order to operate a PET scanner attempts to detect opposing pairs of gamma rays. These gamma rays are produced at the same point within the subject's body, thus they can be used to triangulate a position within this body. ('Prefilter collimator for PET gamma camera', 1997)

The gamma rays are produced from the annihilation of a positron. Positrons are artificially introduced into the subjects body by injecting a radioactive, biologically active molecule, this substance is commonly referred to as a tracer. This tracer is then distributed around the body by the bodies' metabolic processes, thus when the positions of all the annihilated positrons are collated, or the concentration of positrons is found, a 3D picture of the metabolic process should be produced. (Rowe *et al.*, 2008)

When used for clinical diagnosis these 3D images are usually sliced into separate two dimensional (2D) images or slices.

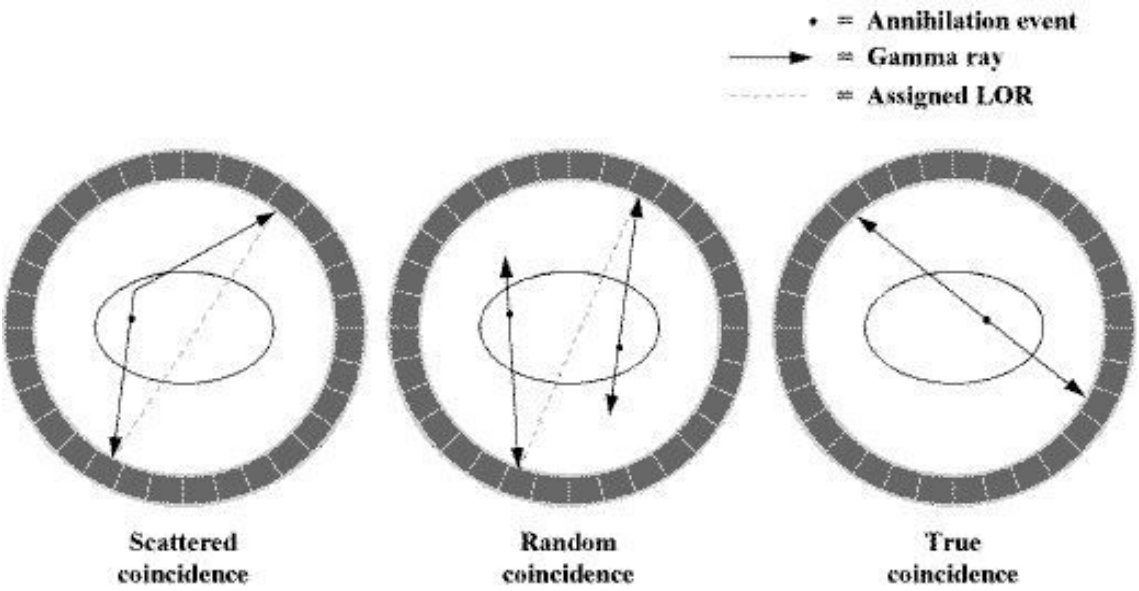


*Illustration 1: This illustration shows an example representing the opposing pairs of gamma rays which are used to determine the location of the positron from the radioactive tracer within the subjects body. (Nandi, 2013)*

**Positron Emission Tomography: Image Reconstruction**

The process by which pairs of gamma ray events are collected and then reconstructed into a 3D image of the concentration of the tracer within the subjects body is known as image reconstruction. Image reconstruction is the process by which an image is estimated from a set of spherical projections of the object in question. These spherical projections are called sinograms, this is because the transform of an off centre object is sinusoidal in these projections. The goal of image reconstruction is to provide accurate slices or images of the concentration of the tracer in the object that is being scanned. (Hudson and Larkin, 1994)

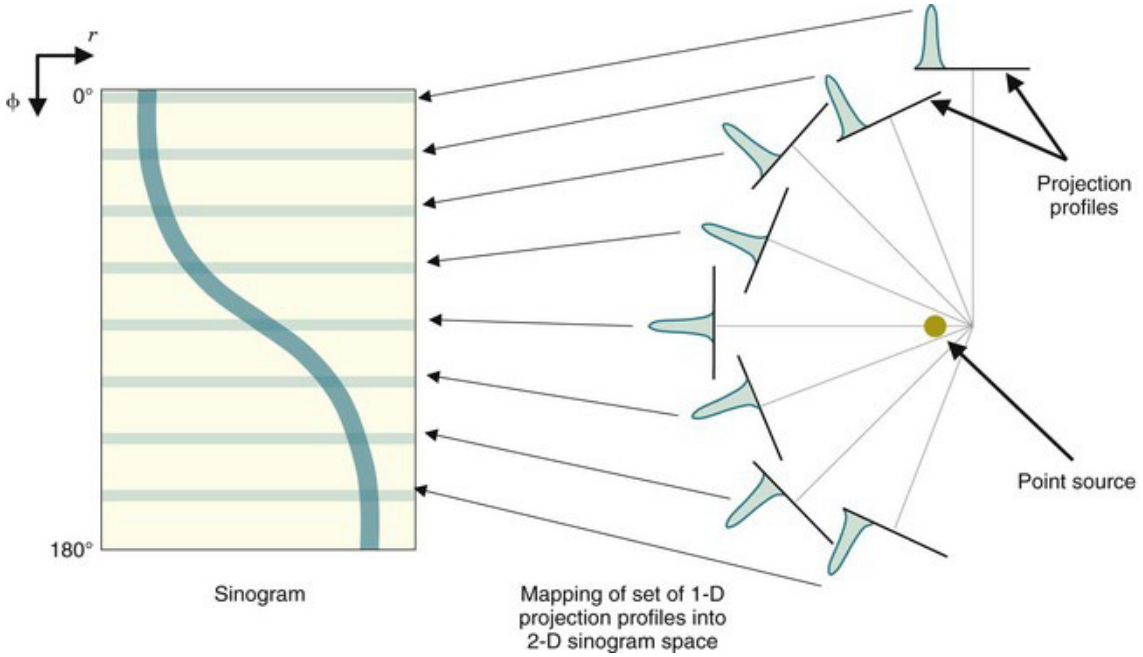


*Illustration 2: This illustration shows an example of how a sinogram is constructed from multiple projections of an object. (Tomographic Reconstruction in Nuclear Medicine | Radiology Key, no date)*

Common image reconstruction algorithms include filtered back projection or the inverse radon transform and iterative reconstruction. (Sagara *et al.*, 2010)

Image reconstruction can also be performed using algorithms such as maximum likelihood expectation maximisation (MLEM) or ordered subset expectation maximisation (OSEM). (Kadrmas, 2004)

Another image reconstruction algorithm is list mode reconstruction, list mode reconstruction is has been designed specifically to aid in motion correction in PET/CT. (Chan *et al.*, 2018)

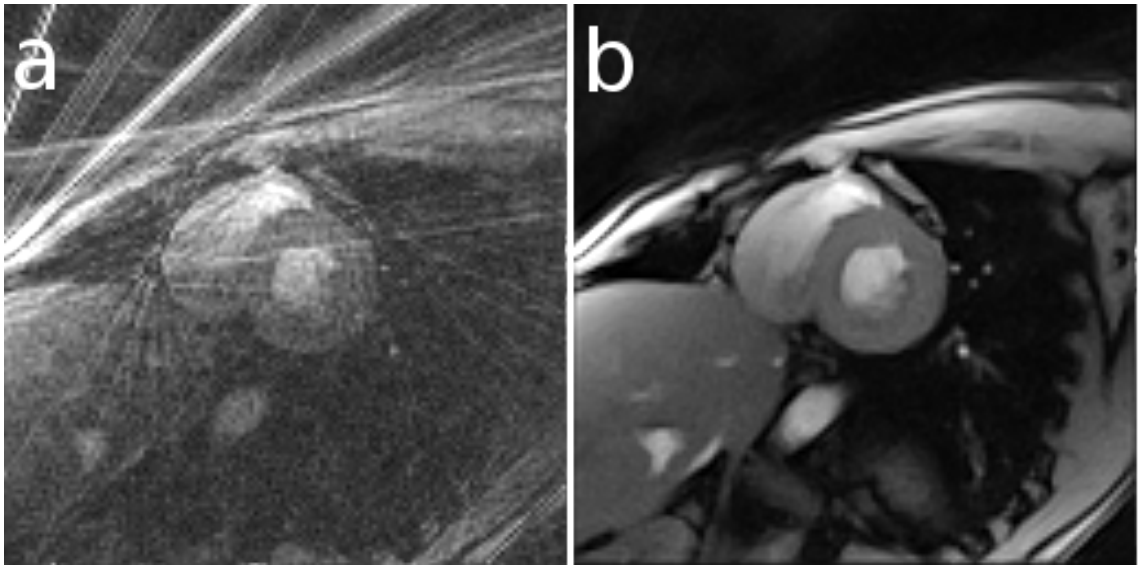


*Illustration 3: This illustration shows an example of an image which has been reconstructed using the filtered back projection or the inverse radon transformation on the left and an image which has been reconstructed using iterative reconstruction on the right. (Heart-direct-vs-iterative-reconstruction.png (PNG Image, 516 × 256 pixels), no date)*

**Positron Emission Tomography: Subject Motion**

Usually when performing a PET scan it can take quite a long time for a substantial enough amount of data to be acquired. A significant amount of data is required in order for the image reconstruction algorithm to be able to generate an output image, otherwise these images may either have inadequate accuracy, contrast or exposure to be useful. Thus a PET scan can take anywhere upwards of 20 minutes to complete. (Townsend, Beyer and Blodgett, 2003) (Conti, 2011)

During a PET scan any movement of the subject will result in the loss of resolution or the introduction of motion related artefacts to the final image, this is similar to the way in which if movement occurs during a long exposure photograph the final image will appear to be blurry. Movement which is known to interfere with PET scans includes, respiratory movement and musculoskeletal movement. (Beyer *et al.*, 2003)

*Illustration 4: This illustration shows an example of subject motion which has occurred during a long exposure photograph. (Mabel Lowman Art: long exposure photography, no date)*

Current standard practice is to attempt to minimise this movement at the source. However, other than minimising the movement at the source there are no standard ways in which to reduce this movement other than to attempt to ignore the artefacts that are caused by this movement. This obviously leads to difficulties when attempting to use these images for clinical diagnosis or research. (Lowe *et al.*, 1995)

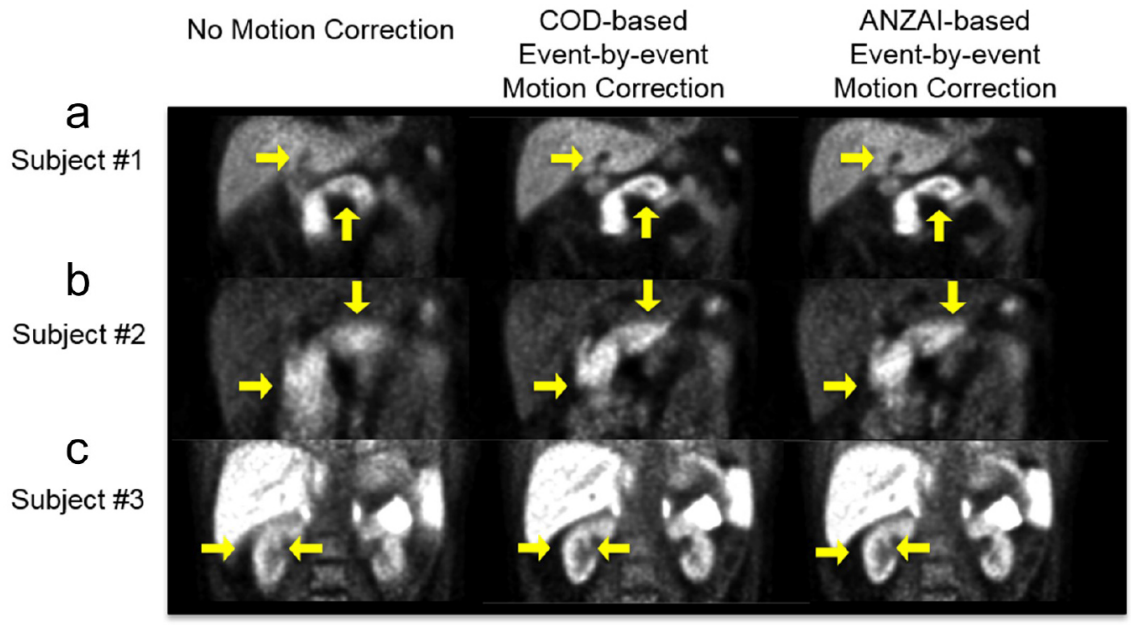


*Illustration 5: This illustration shows an example of PET data which has been affected by the motion of a subject on the left, an example of one form of motion correction on the same PET data in the centre and an example of another form of motion correction on the same PET data on the right. (Ren et al., 2017)*

In order to increase the accuracy of modern PET scanners, additional modalities have been combined with PET in order to aid in the reconstruction process. These additional modalities include, CT (x ray scans) and MR or MRI scans. (Drzezga *et al.*, 2012) (Manber *et al.*, 2015) (Manber *et al.*, 2016) (Picard and Thompson, 1997)

These additional scans are often preformed on the same machine but may require a different bed position as the two parts of the scanner are usually located separately from one another. (Burger *et al.*, 2002)

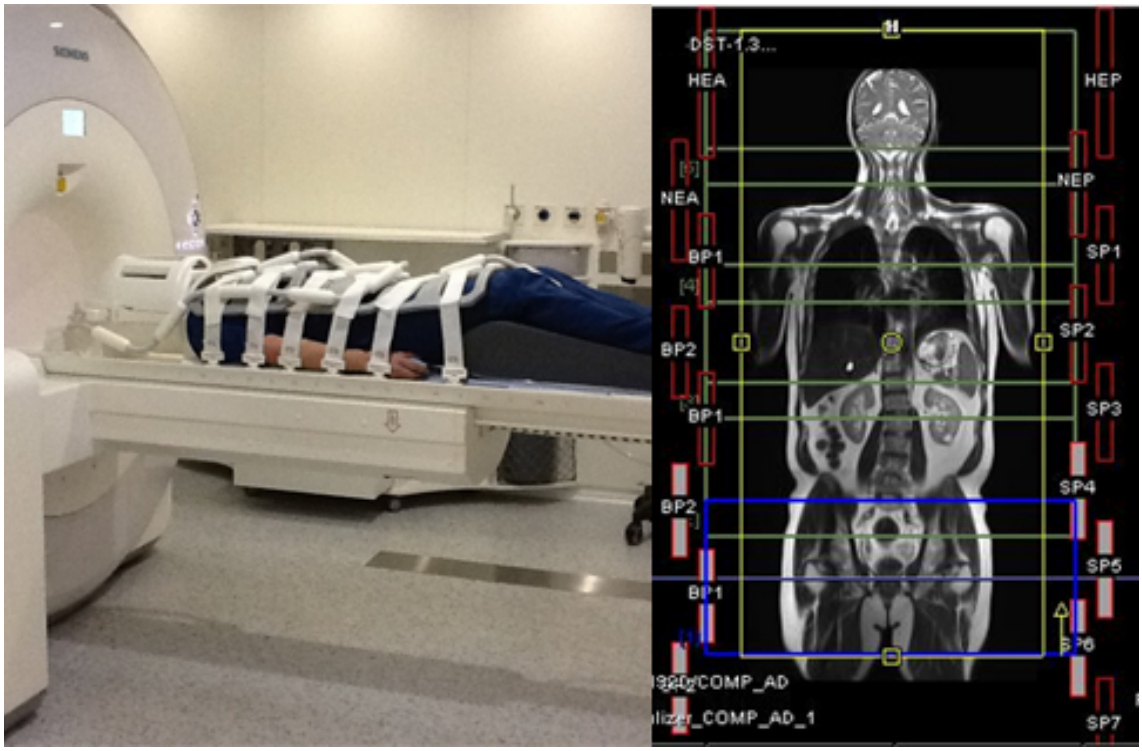


*Illustration 6: This illustration shows an example of a PET/MR scanner on the left and an example of the output of a PET/MR scanner on the right. (New PET MRI scanner is first of its kind in UK, 2012)*

This application will be written specifically to target issues related to subject motion and motion correction on PET/CT scanners. One of the reasons that this application will mainly target PET/CT scanners is that they are usually at an increased risk of being adversely affected by subject movement then PET/MR or other medical imaging modalities. This is because of the way in which the output of the CT scanner is used during the reconstruction of the PET data. (Liu *et al.*, 2009) (Burgos *et al.*, 2014)

Additionally, because this application is being designed to make use of physical motion tracking equipment, it would be unideal to attempt to use this application with a PET/MR scanner. This is because the MR section of the scanner makes use of a large static magnet that not only would damage the electrical components of the motion tracking equipment but could also pull the equipment into the scanner itself, damaging the scanner. (Natarajan *et al.*, 2012)

### Alternative Solutions: Data Driven Motion Tracking

One method of motion correction for PET/CT data is to try and determine the motion of the subject from the data of the PET/CT scan itself, to do this the points at

which the subject is at full inhalation or full exhalation are isolated from the points at which the subject is in the process of inhaling or exhaling. These points are then used to gate or bin the data from the PET/CT scanner into sections where the subject is in motion and where the subject is at rest. (‘Comparison of different methods for data-driven respiratory gating of PET data’, 2013) Thus by using only the data where the subject is at rest, to reconstruct an image, most of the motion related artefacts can be negated. (Bertolli, 2018)

A disadvantage of this solution is that in order for the image, which is reconstructed from the binned PET/CT scan data, to exhibit the same characteristics as a normal PET/CT scan, meaning that for instance there is a reasonable contrast level and that the image is not under exposed, the length of the scan must be increased to make up for the data which is removed in the binning process, where the subject is in motion.

An alternative data driven method of motion correction for PET/CT data is to attempt to extract and use a surrogate or breathing signal to determine the likely position of a subject, this surrogate or breathing signal can then be used to directly warp the PET/CT data in order to attempt to mitigate this signal. The surrogate or breathing signal can either be acquired through the use of a spirometer, a pipe which a subject breathes into and then outputs the velocity of their breath, or through tracking an object attached to the chest of the subject using an optical camera. (McClelland *et al.*, 2017)

A disadvantage of this solution is that additional hardware, such as the spirometer or the camera used to track the object attached to the chest of the subject, and operational steps, such as adjusting the spirometer or affixing the object to be tracked to the subjects chest, are required when performing the PET/CT scan using this motion correction technique. This is an issue as, obviously, hospitals and companies in general are not usually open to the concept of purchasing hardware, which is perceived as optional, where necessary. This is also a concern as radiographers, the people who operate PET/CT scanners and perform scans on subjects, are generally set in their ways and are unlikely to want to change their operating procedure to incorporate the additional steps that this motion correction solution would demand.

Alternatively, a solution has been proposed which attempts to combined the best parts of the two solutions mentioned above. This solution attempts to gate or bin the data into discreet sections that represent each phase of the respiratory cycle. (‘Comparison of different methods for data-driven respiratory gating of PET data’, 2013) This gated or binned data is then used to attempt to determine the likely position of a subject in a similar way to a surrogate or breathing signal, thus this signal can then be used to

directly warp the PET/CT data in order to attempt to mitigate this signal. (Bousse *et al.*, 2017) (Bousse *et al.*, 2016)

An advantage of this method is that because rather than discarding data where the subject is in motion a signal is calculated to describe this motion this method makes use of all of the data that is acquired from the PET/CT scan. This means that a PET/CT scan does not have to be artificially prolonged in order to acquire a usable amount of data.

A second advantage of this method is that it is entirely non invasive and does not require any additional equipment to be performed. This means that because there are no additional optional equipment that must be purchased by hospitals or companies in general, the solution can be widely distributed and used at no extra cost to the scanner operator and at great speed. There are also no additional steps or operating procedures that a radiographer is required to perform during the scanning procedure.

An additional advantage is that because the surrogate or breathing signal is calculated using data from a cross section or slice of the subject, the surrogate or breathing signal reflects the exact internal movement of the subject. This is in contrast to only tracking the surface movement of the subject. As such, this solution is suitable where the internal soft tissue can move independently of the surface movement of the subject, like with respiratory motion.

### Alternative Solutions: Optical Motion Tracking

Another method of motion correction for PET/CT data is to try and track the motion of the subject directly, as a 3D surface, using an optical stereo depth sensing camera, the process of which is as follows.

To temporally synchronise the acquisition of frames the optical stereo depth sensing camera is triggered through its general purpose input output (GPIO) port using an Arduino. The Arduino microcontroller is programmed to generate a regular pulse from between twenty five to fifty cycles per second, this signal is then also connected to the PET/CT scanner through its gate input and indirectly coupled to the optical stereo depth sensing camera trigger input using an optocoupler, an optocoupler is a type of integrated circuit which is used to transfer electrical signals between two isolated circuits by using light.

To specially synchronise the cameras the position of the optical stereo depth sensing camera is then acquired relative to the position of the PET/CT scanner using the

surface normals of a collection of tetrahedrons. The tetrahedrons are affixed to the outer bore of the PET/CT scanner such that at least three of each tetrahedron's faces are always visible from the perspective of the optical stereo depth sensing camera. The surface normals of the tetrahedrons are positioned as such that the intersection point of the surface normals of all tetrahedrons intersect at the origin point of the PET/CT scanner. This can then be used to calculate the origin and orientation of the PET/CT scanner in relation to the position of the optical stereo depth sensing camera.

To further establish the transformation from the position of the optical stereo depth sensing camera to the PET/CT scanner multiple scans of a radioactive point source are taken simultaneously by both the optical stereo depth sensing camera and the PET/CT scanner. A homogeneous transformation is then calculated from the PET/CT scanner to the optical depth sensing camera using least squares regression, this finds a homogeneous transformation which attempts to minimise the distance between two sets of data.

Least squares regression finds this homogeneous transformation by attempting to determine a line of best fit through a data set, this is achieved by minimizing the distance from a line of best fit to each point in a data set represented by the sum of the square of the distance from each point in the data set to the regression line. (Geladi and Kowalski, 1986)

Because a majority of the calculations involved with motion tracking and registration are relatively time consuming, the depth images are output to storage together with a header representing the specific settings used for that particular acquisition, then the motion tracking and registration can be performed after the acquisition has finished, this means that the motion tracking and registration calculations are performed at a separate time and in one block rather than in real time as the acquisition is performed. Additionally, to save on storage space and computation time pixels with invalid depth values are set to a NAN value and then removed.

Once the depth images have been reloaded, after the initial acquisition, and converted into point clouds all the remaining calculations are performed using the open source PCL.

Firstly, in order to further reduce the time spent on the motion tracking and registration sections of the application the point clouds, which have just been calculated, are downsampled from their original resolution to a lower resolution.

Secondly, in order to ensure that the ROI is tracked, rather than a larger region which may not share the same motion as the ROI and is thus irrelevant, like the bore of the PET/CT scanner, points like the ones representing the object of the bore of the PET/CT

scanner must be segmented and removed. To do this, a plane is defined using the random sample consensus (RANSAC) algorithm to find the position of the PET/CT scanner bore plane, because there are a large number of points which are used to represent the PET/CT scanner bore the RANSAC algorithm should need very few iterations to converge on the object itself, this is because the RANSAC algorithm works best when given a data set where there are points which are both clustered as part of a region and points which are not.

RANSAC uses a kind of voting system where points in the data set vote for which line or plane of best fit represents the data set as a whole, this works because it is assumed that the points which are not part of the cluster in question will not vote for any one line or plane of best fit consistently, rather voting randomly and not for the same line or plane of best fit as each other, and that the points which are part of a cluster will vote for the same line or plane, eventually drowning out the other votes. (Fischler and Bolles, 1981)

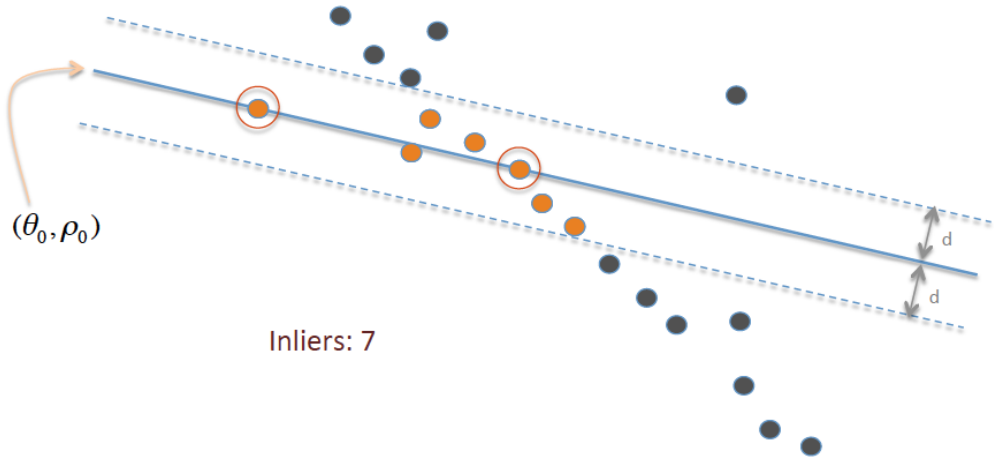


*Illustration 7: This illustration shows an example of how the RANSAC algorithm converges on a line or plane of best fit. (RANSAC_Inliers_and_Outliers.png (PNG Image, 986 × 460 pixels), no date)*

Once the plane of best fit for the bore of the PET/CT scanner is found all the points in the point cloud which fall within a set distance of this plane can be removed in order to remove the bore of the PET/CT scanner from the point cloud as a whole.

Before the motion tracking and registration procedure can begin a reference scan of the PET/CT scanner is taken in order to be used later as a subtractive point cloud in the event that a part of the scanner cannot be clustered and removed.

In order to segment the point cloud further into the ROI and the tetrahedrons, used to find the position of the optical stereo depth sensing camera, the Euclidean clustering algorithm is used.

The Euclidean clustering algorithm works by firstly defining one or many seed points for the clustering algorithm to start from, then for each seed point the algorithm iteratively finds every point that is within a set radius of the initial points, if a point is within a set radius of the initial point it is added to that cluster and the process is repeated for the new points in the cluster, this continues until there are no more points which are within the distance specified of the points in the cluster. The Euclidean clustering algorithm works in a similar manor to segmentation algorithms like region growing segmentation. (Jain, Murty and Flynn, 1999)

The ROI is then selected manually, by clicking on the cluster in question, from the output from the Euclidean clustering algorithm.

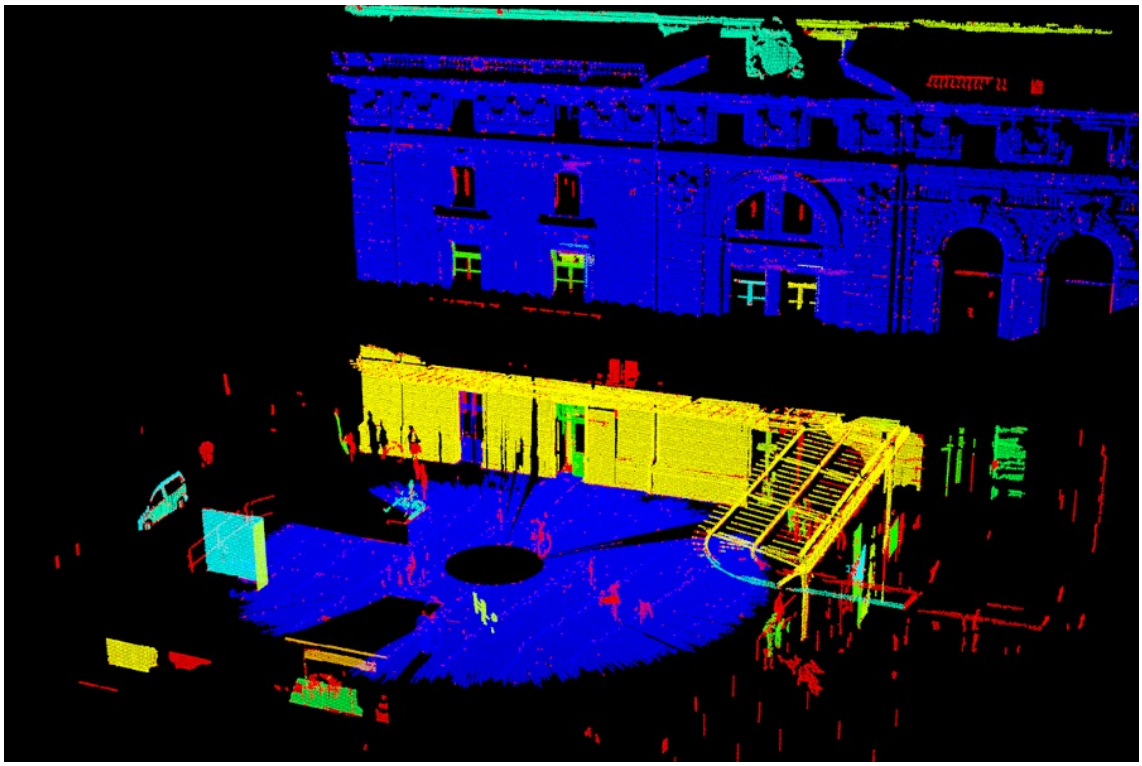

*Illustration 8: This illustration shows an example of how the Euclidean clustering algorithm segments a point cloud into a series of clusters, each differently coloured point cloud represents a different cluster. (Documentation - Point Cloud Library (PCL), no date e)*

Because there are sections of the ROI which are occluded in each individual frame, meaning that they are not visible to the optical stereo depth sensing camera, point clouds from several subsequent frames are combined to give a more accurate point cloud of the cluster in question. The iterative closest point (ICP) algorithm is used to register each individual point cloud from each frame into one cohesive model of the ROI. (Besl and McKay, 1992)

The output of the ICP algorithm is then passed through a filtering algorithm which removes points which fall outside of the standard deviation of the point cloud as a whole.

The moving least squares surface reconstruction algorithm is then applied to the point cloud in a similar way to the least squares regression algorithm, however, rather attempting to find a line of best fit through a data set, the moving least squares surface reconstruction algorithm attempts to interpolate between the points in a data set in order to approximate missing data and leave a uniform surface. (Alexa *et al.*, 2003)

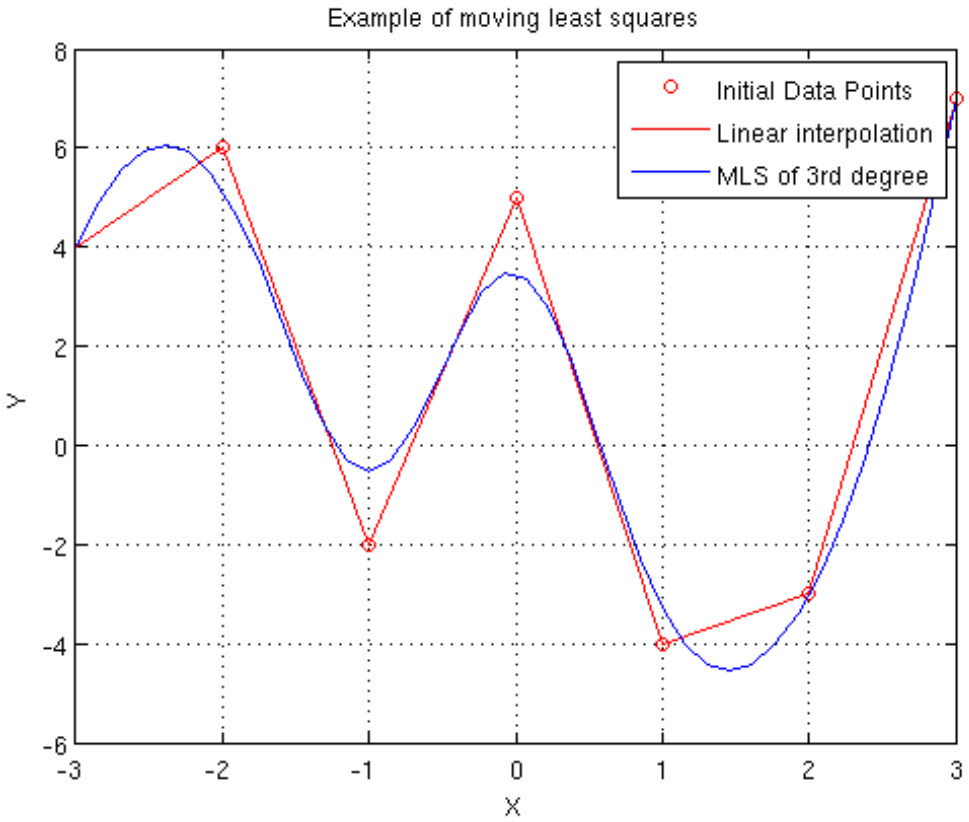


*Illustration 9: This illustration shows an example of how the moving least squares surface reconstruction algorithm interpolates points on a 2D data set. (Moving_Least_Squares2.png (PNG Image, 560 × 420 pixels), no date)*

The model which is output from the filtering algorithms is then edited by hand to remove any remaining outlying points. Structures which may also move independently of the ROI as a whole, for instance the limbs of a subject, are also removed in this step in order to increase the accuracy of the registration algorithm later.

In order to find the difference between the positions of the ROIs, as the acquisition progresses, the transformation between the ROIs are then calculated using the ICP registration algorithm.

In order to accelerate and increase the accuracy of the ICP registration algorithm an initial guess transformation is passed to the method at its start time, this guess transformation can be set to the output transformation of the ICP registration step of the previous frame. If, for whatever reason, the ICP registration step of the previous frame was unsuccessful or the error is determined to be too great, the initial guess transformation is set to a new guess transformation determined from the difference between the centre of gravities of the source point cloud and target point cloud.

Once the transformation is calculated using the ICP registration algorithm a distance or error score is also calculated for each set of point cloud pairs, this distance or error score is equal to the sum of the mean square distance between all the points of the source point cloud, after it has had the transformation applied to it that was found in the previous step, and the target point cloud. This distance or error score increases as the difference, defined as the proportion of the two point clouds which are not overlapping, between the source point cloud and the target point cloud increases. If the distance or error score is larger than a set threshold the transformation between the source point cloud and target point cloud is deemed to be inaccurate and it is excluded from the output.

The final output from the application is a list of the valid transformations determined in the previous step. These transformations can be used, in conjunction with the timestamps of the depth images and the transformation from the optical stereo depth sensing camera to the PET/CT scanner, to determine the amount that each frame in the PET/CT scanners output needs to warped by in order to attempt to remove the motion of the subject. (Miranda *et al.*, 2017) (Noonan *et al.*, 2015)

A disadvantage of this solution is that it only calculates the amount to transform by, to remove the perceived motion, based on the surface movement of the subject. As such, this solution is only acceptable when being used to track objects where the internal structure is rigid, like the head. This solution is is not suitable where internal soft tissue can move independently of the surface movement of the subject, like with respiratory motion.

An additional disadvantage of this solution, like with some of the solutions mentioned above, is that additional hardware, such as the optical stereo depth sensing camera used to track the motion of the subject, is required when performing the PET/CT scan using this motion correction technique. This is an issue as, obviously, hospitals and companies in general are not usually open to the concept of purchasing hardware which is perceived as optional where necessary.

An advantage of this solution is that the domain is far more mature than the data driven motion tracking solutions. As such, there is a larger breadth of research, from multiple different disciplines, which has already been carried out in this area, thus a solution which draws from this research may be easier to implement with a higher likelihood of guaranteed results.

Because of this the solution to motion correction in PET/CT discussed in this dissertation will attempt to implement an application which draws from the research in optical motion tracking mentioned above.

## 3.2 Comparison of Technologies

### Microsoft Kinect Camera: Overview

In order for an application to be written that would be able to track the motion of an object in a scene some kind of camera which can measure the distance to points in a scene must be used, an example of a cheap, consumer, off the shelf depth sensing camera would be the Kinect camera.

The Kinect camera is a family of depth sensing cameras developed for use with the Xbox line of game consoles by Microsoft. Each camera consists of, at least, an RGB camera for capturing RGB video data, a depth sensor for capturing depth images, which can be converted into point cloud data, and a multi array microphone for capturing sound data. Together these devices can, through software, provide 3D motion capture capabilities. (Zhang, 2012)

The maximum distance to which both versions of the Kinect camera can operate to is adjustable out to approximately four to eight metres, this is because the intensity of the infrared emitter is not great enough to be perceived much beyond this distance, the intensity of the infrared emitter is calculated based on the maximum distance in a scene.

The minimum distance to which both versions of the Kinect camera can operate to is approximately around half a metre, this is because at distances smaller than half a metre the infrared receiver is over saturated and cannot accurately read the distance for a given pixel of the depth sensor. However, the minimum distance, to which the Kinect camera can operate to, can be circumnavigated. This can be achieved using the Kinect camera version two, there is a version of the Kinect camera firmware, which can be flashed to this version of the Kinect camera, that greatly reduces the average intensity of the infrared emitter. This means that while the maximum distance that the Kinect camera can operate to is greatly reduced so is the minimum distance. Using this method the minimum distance to which the Kinect camera can operate to can be reduced to as little as up to a few centimetres away from the Kinect camera's aperture. Such a small minimum operating distance is useful as it allows for the Kinect camera to be placed closer to the subject of a scan, this means that a greater amount of a given depth image is taken up by the ROI rather than by objects surrounding the ROI, thus the scan of the ROI will be of a higher resolution and by extension the registration of the point cloud of the ROI will be more accurate later. (Noonan *et al.*, 2015)

Both versions of the Kinect camera use a different type of technology for their depth sensing capabilities, however, what does remain the same is that both types of technologies operate using some kind of infrared emitter, which produces an infrared

laser, and an infrared receiver. Both depth sensors use their infrared emitter to project a pattern of lines or points onto a surface which is then, in turn, reflected from the surface into the infrared receiver of the Kinect camera. This pattern of infrared points or lines is then used, in a different fashion by both cameras, to determine the distance from the Kinect camera to each pixel of a depth image in space that is in view of the camera. The depth image which is constructed from the data from the Kinect camera can also then be registered to an RGB image which is captured, simultaneously, from an additional complementary metal oxide semiconductor (CMOS) camera, located on the Kinect camera. This can aid in the creation of a full 3D scene later. (*Color supported generalized-ICP*, no date)

The full specification for both the Kinect camera version one and the Kinect camera version two can be seen in the appendix below. (Appendix K:) (Noonan *et al.*, 2015)


*Illustration 10: This illustration shows an example of the Kinect camera version one on the left and an example of the Kinect camera version two on the right. (Smeenk, 2014)*

**Microsoft Kinect Camera: Structured Light Sensor**

The Kinect camera version one uses a type of technology for its depth sensing capabilities known as a structured light depth sensor. A structured light depth sensor works by using it's infrared emitter, which is offset in the horizontal plane from the

centre of the camera, to project a pattern of alternating bright and dark vertical lines onto a given scene.

After having reflected off of the scene, this pattern of alternating bright and dark vertical lines is then detected by the infrared receiver, which is also offset in the horizontal plane from the centre of the camera but in the opposite direction to the infrared emitter.

Because the infrared receiver is offset in the horizontal plane from the infrared emitter the pattern of alternating bright and dark vertical lines, which is received by the infrared receiver, is deformed by the difference in distance that it has had to travel before intersecting with an object, this can be explained as the perspective of the scene which has been produced by offsetting the infrared emitter and the infrared receiver from one another in the horizontal plane. This deformation occurs and can be measured in a similar way to how the brain can measure depth from the input images of the two eyes, these two eyes are also offset in the horizontal plane from one another. Simply, the depth of an object in the scene can be estimated, by the Kinect camera, based upon the amount of deformation of the alternating bright and dark vertical lines at any one point in the image which is received by the infrared receiver. (Khoshelham and Elberink, 2012)

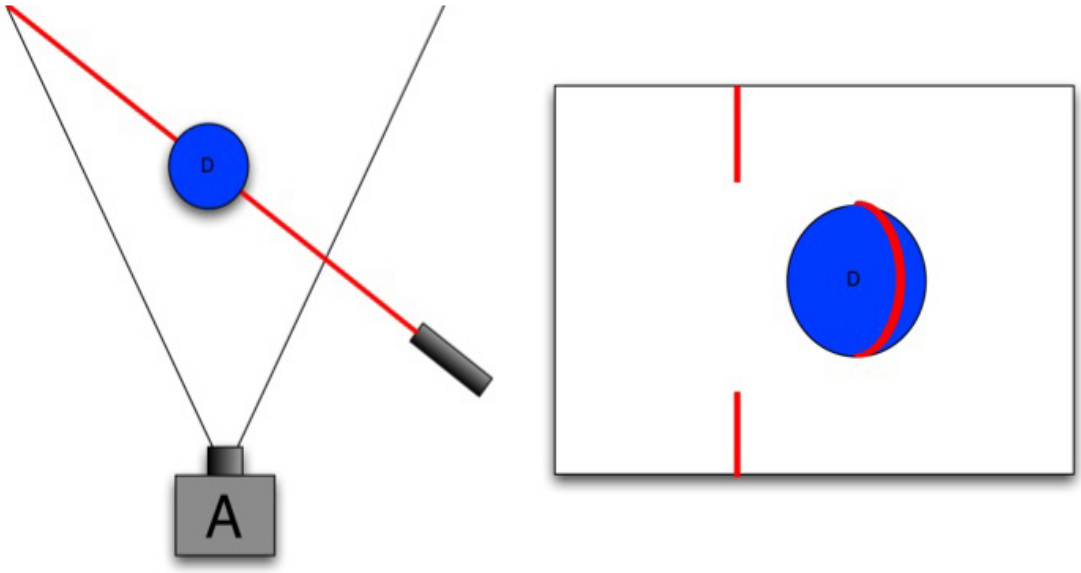


*Illustration 11: This illustration shows an example of a scene where an infrared laser is projecting one vertical line onto a sphere, the deformation of this vertical line can then be seen from the perspective of the camera. (Lau, 2013)*

A disadvantage of this solution is that because the infrared emitter is offset in the horizontal plane from the infrared receiver there will be a black outline around the edges of any objects in the scene, where a distance measurement cannot be made.

This is inherent in the design of the structured light depth sensor and is caused because of the perspective of where the infrared emitter is located in relation to the infrared receiver. One way to think about this is that different parts of the scene are occluded by different parts of the objects in the scene, this is in the same way that if a person were to hold their thumb in front of their face and look with their left and then their right eye their thumb would occlude different areas in their field of view (FOV).

### Microsoft Kinect Camera: Time of Flight Sensor

The Kinect camera version two uses a type of technology for its depth sensing capabilities known as a time of flight (TOF) depth sensor. A TOF depth sensor works by using it's infrared emitter to regularly pulse a pattern of infrared pixels or dots onto a given scene.

After having reflected off of the scene, the pulse of infrared pixels or dots is then detected by the infrared receiver.

First the time that the infrared pixel or dot was in flight must be found. To find the time that the infrared pixel or dot was in flight the time at which the infrared pixels or dots were created must be taken away from the time at which the infrared pixel or dot was received, the result of which must then be divided in two to find the true TOF. This can be done because the time at which each pulse of infrared pixels or dots are created is known and recorded. The sum of the time at which the infrared pixels or dot were created and the time at which the infrared pixel or dot was received must be divided in two because otherwise the TOF calculated would represent the full round trip from the infrared emitter to the object and then back to the infrared receiver, thus because the trip is the same length from the infrared emitter to the object as it is from the object to the infrared receiver the total TOF can be divided in two to find just the TOF of one half of this trip.

Then, the distance from the object, that the infrared pixel or dot was reflected off of, to the infrared receiver can be found by integrating for the distance using the velocity of the object multiplied by the time that it was travelling for. Because the velocity of infrared light is known and the light neither accelerates or decelerates this is a very simple solved equation. (Lindner, Schiller and Koch, 2010)

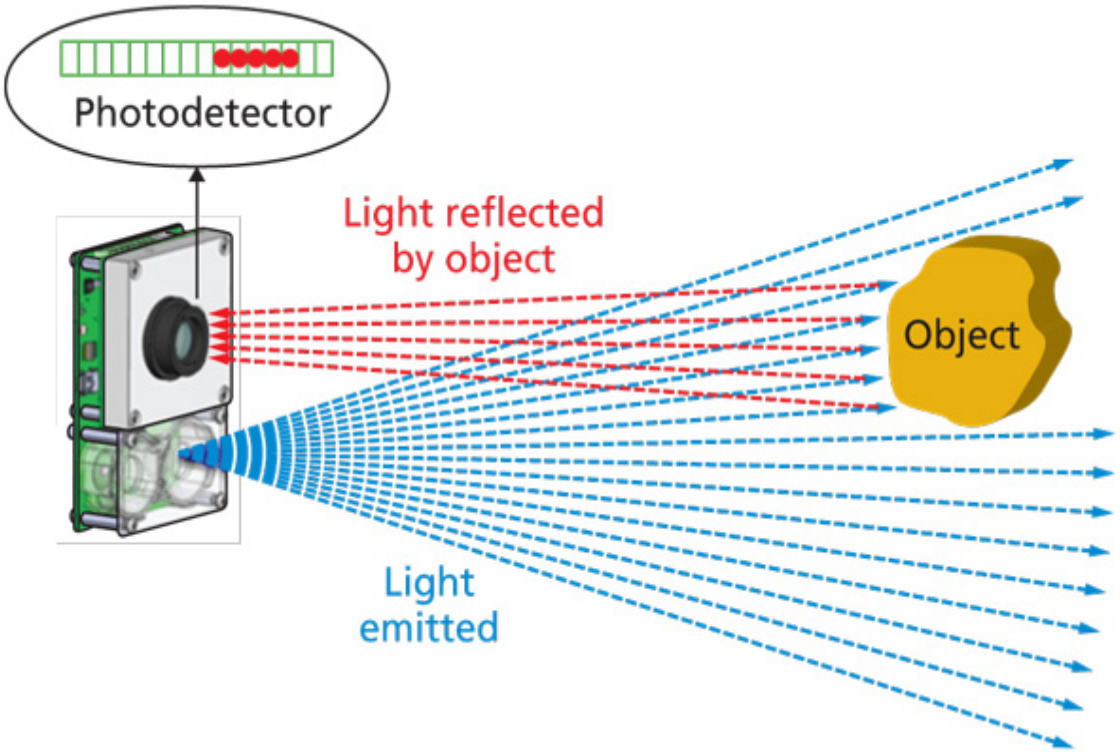


*Illustration 12: This illustration shows an example of a scene where a infrared laser is illuminating an object, the reflected light can then be seen returning to the camera. (Poulin, 2014)*

This type of technology for depth sensing is very similar to the way in which a surveying device known as a LiDAR scanner works. One of the only differences between a Kinect camera version two and a LiDAR scanner is that rather than pulsing many infrared pixels or dots at once a LiDAR scanner will wait for an infrared pulse to return before moving to acquire another distance reading. The method that a LiDAR scanner uses is far more accurate than scattering a scene with an infrared pulse, however it is far too slow to be used in real time.

The application produced in this project will initially make use of the Kinect camera version one, thus it is important to understand the technologies through which the camera operates in order to be able to produce a solution which is of the highest quality possible. However, the application produced in this project will be written in a modular fashion so that the Kinect camera version two, or any other kind of stereo depth sensing camera, could be added in the future with as few compatibility issues as possible.

## Scanner Synchronisation: Overview

In order for the depth images and point clouds, which are acquired from the Kinect camera, to be relevant to the output of the PET/CT scanner and to then utilise the Kinect camera to track the motion of the subject during a PET/CT scan, the Kinect camera must be temporally and spatially synchronised to the PET/CT scanner. In order for this to be achieved the output from the depth sensing camera must have some relation to the PET/CT scanners output, thus this means that both the time at which each frame is captured and the position where each frame is captured from on both the optical depth sensing camera and the PET/CT scanner must be recorded.

The act of temporally and spatially synchronising the Kinect camera to the PET/CT scanner is necessary so that the correct frames are used to warp or otherwise affect the PET/CT data and so that the correct areas of the correct frames are used to warp or otherwise affect the correct areas of the correct PET/CT data.

## Scanner Synchronisation: Temporal Synchronisation

One method through which the the time at which each frame is captured could be made relative, from the depth sensing camera to the PET/CT scanner, would be by injecting a common clock pulse into both scanners. To inject a common clock pulse into both scanners something which produces a regular pulse must be produced, usually a microcontroller like an Arduino is used to output a regular pulse from between twenty five to fifty cycles per second, this is similar to the way in which in music technology electronic instruments like drum machines and synthesisers are synchronised to each other using a common trigger. This pulse is then used internally by both scanners to set the time that an acquisition is made to the pulse of the microcontroller.

A disadvantage of this solution is that it is very unlikely that a given PET/CT scanner would have ports that would allow for the injection of a clock pulse. For instance the Sedecal Argus preclinical PET/CT Scanner, which is available for the testing of the solution produced by this project, does not have the ability for its clock to be set externally. This would dramatically limit the scope and availability of this solution. Additionally, it is unlikely that most hospitals or companies in general would allow for a port to be added to such an expensive piece of equipment as a PET/CT scanner.

Alternatively, the time at which each frame is captured could be recorded and used to order the scan data, from both the depth sensing camera and the PET/CT scanner, so that each frame from one scanner could be assigned to the closest frame, temporally, from the other scanner. This could be achieved by storing the motion data from the optical depth sensing camera with a timestamp which reflects when the data was

acquired. This timestamp can be used to gate or bin the PET/CT data. (‘Comparison of different methods for data-driven respiratory gating of PET data’, 2013)

A disadvantage of this solution is that the express permission and access of the PET/CT scanner manufacturer would be required for it to work. This is because PET/CT scanners do not by default output the types of timestamps that would be required for this solution to work, so access to the application programming interface (API) of the specific PET/CT scanner in question would be required in order to request these timestamps.

Another method through which the time at which each frame is captured could be made relative would be to place a rotating radiation point source into the PET/CT scanner, the position of this rotating radiation point source could be used to ensure that the depth sensing camera was using the same frame as the PET/CT scanner. This would work as it would be expected that the rotating radiation point source would be located in the same place in both frames.

An advantage of this solution is that it could be implemented without the express permission and access that would be required by any of the other solutions mentioned above. For instance, permission to irreversibly damage the PET/CT scanner while adding clock inputs or access to the API of the PET/CT scanner to request timestamps would not be required.

A disadvantage of this solution is that there are no sources to back up the validity of this solution. For instance, it is very likely that the resolution of the scan that the PET/CT scanner would be able to acquire in the short space of time before the radiation point source rotated to another position would be too low to be able to use it to accurately temporally synchronise the depth sensing camera to the PET/CT scanner.

### Scanner Synchronisation: Spatial Synchronisation

The objective of spatially synchronising the Kinect camera to the world around it and then to the PET/CT scanner itself is paramount to the operation of the application as a whole.

First the calibration parameters, which correct for the perspective of the Kinect camera must be found. Then the transform that will translate the origin of the Kinect camera’s coordinate system to the origin of the PET/CT scanners coordinate system must be found. This is important for a number of reasons.

For one, the initial depth images and by extension the point clouds, which are calculated from the depth images, must reflect accurately the objects in the scene, otherwise all subsequent calculations will be inaccurate.

Second, the position that each depth image corresponds to must be known otherwise the output from the application will not be able to be applied to the PET/CT scan data. This is true so that the motion data gathered, by the application, is applied to the correct part of the PET/CT scan data.

First, the Kinect camera must be calibrated so that its output is mapped to a known measurement system, for instance, so that its output is described accurately in millimetres.

To find the correct perspective and to remove distortion from the final depth image, that a Kinect camera produces, the output of a Kinect camera must be compensated by some form of calibration of the measuring equipment. A mathematical model is used to describe the way in which to transform each depth value in a depth image to a point cloud. There are values in the mathematical model which can be used to adjust the perspective of the transform, such as the offset from the aperture of the Kinect camera to the Kinect camera sensor and the focal length of the Kinect camera, this is to take into account the optical aberrations associated with using optical lenses.

The mathematical model to transform a depth image to a point cloud is essentially based on the same simple principle as a pinhole camera, this is where each point on an object has one ray of light which can pass through the pinhole, of the Kinect camera, and represent it in the final image. (*Imaging Information - OpenKinect*, no date)

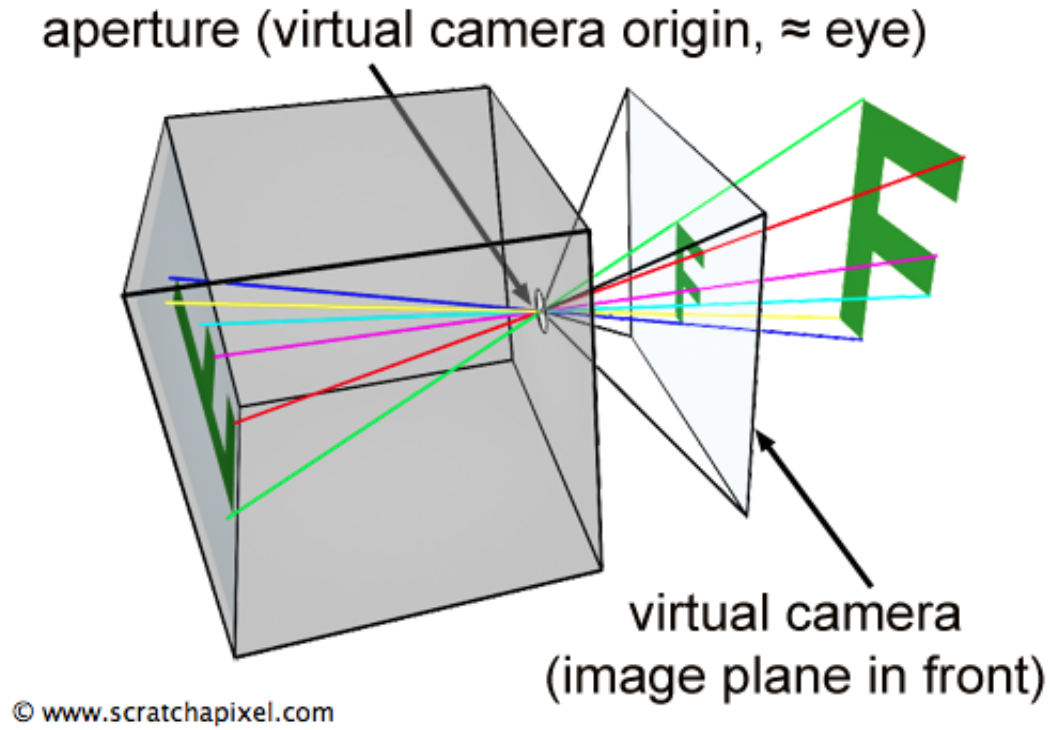


*Illustration 13: This illustration shows an example of how the principle of a pinhole camera works. (3D Viewing: the Pinhole Camera Model (A Virtual Pinhole Camera Model), no date)*

This calibration process is usually performed using patterns specifically designed for the purpose of calibrating depth sensing cameras. One such crude method to calibrate the depth sensor of a Kinect camera involves measuring the perceived distortion of spherical objects at different distances from the Kinect camera and then using the perceived distortion of these spherical objects to calculate a calibration which will remove as much of this perceived distortion as possible. (Staranowicz *et al.*, 2015)

After the parameters to calibrate the Kinect camera to the world have been found a translation from the Kinect camera to the PET/CT scanner must be found. One method to find a translation from the Kinect camera to the PET/CT scanner is to use shapes which have been placed in such a way onto the front surface of the PET/CT scanner that they point to the origin of the PET/CT scanner. This method is mentioned above in the alternative solution using an optical depth sensing camera, in the method mentioned above regular tetrahedrons are placed onto the front surface of the PET/CT scanner in such a way the the normals to the surfaces of the tetrahedrons, when extrapolated out to infinity, intersect at the origin of the PET/CT scanner. (Miranda *et al.*, 2017)

A disadvantage of this solution is that the accuracy of where the origin of the PET/CT scanner is calculated as being located is entirely reliant upon the accuracy of where the shapes or tetrahedrons have been placed. Because these shapes or tetrahedrons will have to be placed onto the front surface of the PET/CT scanner by hand it is very likely that there will be a high degree of error in the position and rotation with which they are placed.

Another method to find a translation from the Kinect camera to the PET/CT scanner is to scan an arrangement of radiation point sources. These radiation point sources can be scanned simultaneously by both the Kinect camera and the PET/CT scanner and then registered to each other in order to find the homogeneous transformation that represents the difference between the origins of both the Kinect camera and the PET/CT scanner. For this solution to function in all dimensions there must be at least three radiation point sources to track the horizontal, vertical and depth dimensions, an extra point source should also be included for redundancy and error checking.

An advantage of this solution is that regardless of the positions of the radiation point sources, unless they are occluding themselves from either the Kinect camera or from the PET/CT scanner, a transformation will always be able to be found accurately. This is because, regardless of the position of the radiation point sources, the radiation point sources will, by their very nature, always contain the horizontal, vertical and depth translations in the relation of their positions to each other.

### Qt Graphical User Interface: Overview

Qt graphical user interface (GUI) (*qt/qtbase: Qt Base (Core, Gui, Widgets, Network, ...)*, 2018) is an open source framework for creating cross platform native GUIs in C++ applications.

The application produced in this project will make use of the Qt GUI in order to develop its front end. Qt has been selected for this role as it is by far and away the most popular C++ cross platform GUI toolkit. (Eng and Eirik, 1994) (*C++ GUI Programming with Qt4 - Jasmin Blanchette, Mark Summerfield - Google Books*, no date)

There are very few alternative tool kits that can be use to create cross platform GUIs, GTK+ is an example of one of the only other cross platform GUI tool kits written in either the C or C++ programming languages. The main disadvantage of GTK+ is that it is inherently linked to the GNU's not Unix (GNU) network object model environment

(GNOME) desktop environment, if a program written with a GTK+ GUI is to be loaded on a none GNOME operating system then a majority of the GNOME desktop environment must be included and loaded with that application. This is obviously a disadvantage as including the GNOME desktop environment as a dependency of an application increases the size of that application significantly and loading the GNOME desktop environment with the application has a significant overhead where the application will use large amounts of both memory and processor time only on the GNOME desktop environment. (*Running GTK+ Applications: GTK+ 3 Reference Manual*, no date)


*Illustration 14: This illustration shows an example of the popularity of the Qt GUI toolkit verses the GTK+ GUI toolkit overtime, Qt is represented by the blue trend line and GTK+ is represented by the red trend line. (Qt, GTK+ - Explore - Google Trends, no date)*

### OpenKinect Libfreenect Driver: Overview

OpenKinect Libfreenect (*OpenKinect/libfreenect: Drivers and libraries for the Xbox Kinect device on Windows, Linux, and OS X*, 2018) and OpenKinect Libfreenect2 (*OpenKinect/libfreenect2: Open source drivers for the Kinect for Windows v2 device*, 2018) are open source drivers for the Kinect camera version one and the Kinect camera version two respectively, written in the C and C++ programming languages. These drivers can be used to interface directly with the Kinect camera version one and Kinect camera version two in order to perform a multitude of tasks, including, creating a data connection between application and Kinect camera, moving the motor of the

Kinect camera, requesting depth images from the Kinect camera and requesting RGB images from the Kinect camera.

The application produced in this project will make use of the OpenKinect Libfreenect and OpenKinect Libfreenect2 drivers as an interface to the Kinect camera version one and the Kinect camera version two respectively. The application produced in this project will specifically use the OpenKinect Libfreenect and OpenKinect Libfreenect2 drivers as a way to create a data connection between the application and the Kinect camera, to control the motor of the Kinect camera and access the accelerometer of the Kinect camera and as a way to retrieve depth and RGB information from the Kinect camera.

OpenKinect Libfreenect and OpenKinect Libfreenect2 will be used as the drivers to access the Kinect cameras the official Microsoft drivers. The reason that the OpenKinect Libfreenect and OpenKinect Libfreenect2 drivers will be used is because these drivers are entirely open source and cross platform, because of this the OpenKinect Libfreenect and OpenKinect Libfreenect2 drivers will not hinder the ability for the application produced in this project to be able to be compiled on nearly every operating system, whereas the official Microsoft drivers will only run on operating systems produced by Microsoft. The official Microsoft drivers also do not offer any additional functionality which may make these drivers more appealing.

### Point Cloud Library: Overview

In order for a solution to be able to save and load point clouds and for a solution to be able to perform calculations such as the de noising and registration of point clouds a library would have to be found that provides these functions. An example of such a library would be the PCL. The PCL (*PointCloudLibrary/pcl: Point Cloud Library (PCL)*, 2018) is a cross platform open source framework facilitating the creation, storage, processing and visualisation of point clouds.

A point cloud is an arbitrarily ordered or unordered set of Cartesian points in space. An ordered point cloud directly represents an image or matrix where the rows and columns act similarly to the pixels of an image. For instance, the output from a stereo depth sensing camera or a TOF camera would usually fall into this category. The advantages of using an ordered point cloud is that because it is certain that the adjacent points in a point cloud are in fact the neighbours of that point any algorithms that call for knowledge of the neighbours of a point will obviously execute at a greater speed because they themselves do not have to calculate the neighbours of each point.

One of the most common use of point clouds is as the standard output of surveying devises such as LiDAR scanners.

The PCL is capable of processing that includes, the cleaning and removal of extraneous or outlying points, the registration of one point cloud to another, the segmentation of one point cloud into two or more parts and mesh reconstruction from point clouds. The PCL is supported and maintained by quite a large proportion of the technology industry including, Nvidia and Toyota. (*PCL Developers Blog*, 2015)

The application produced in this project will make use of the PCL as a way to save and load point clouds which have been calculated from the depth images received from the Kinect camera. The application produced in this project will also make use of the PCL as a way to clean point clouds, register point clouds and track point clouds.

It is advantageous to use the algorithms which are present in the PCL rather than writing them from scratch as the algorithms present in the PCL have been optimised and tested to a high standard and as such it is more likely that a high quality solution could be produced in a shorter time frame.

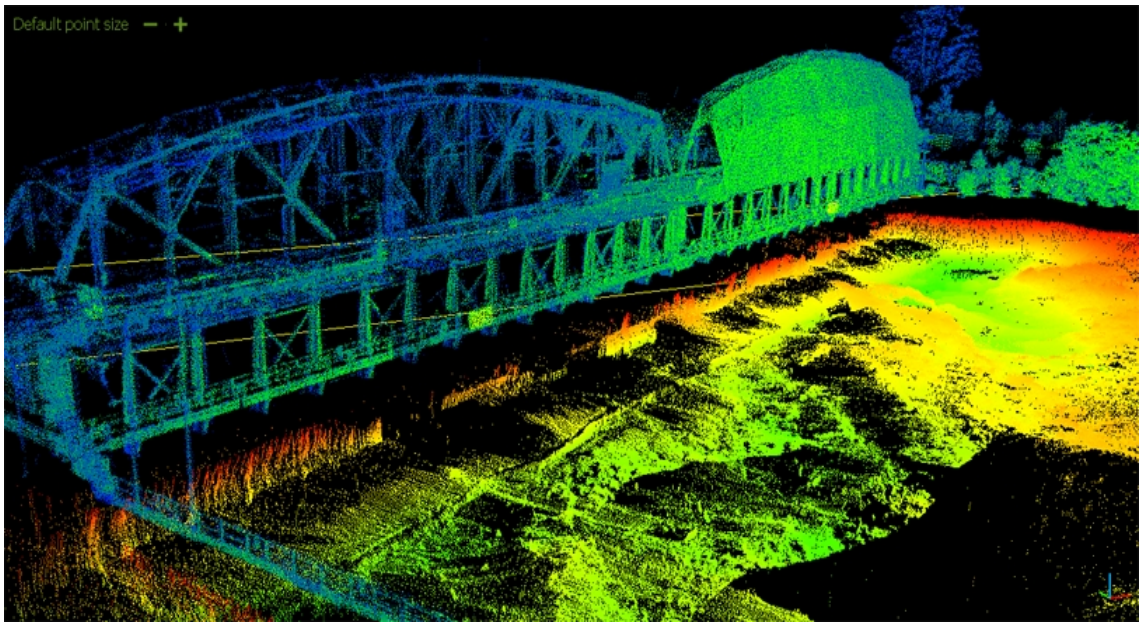


*Illustration 15: This illustration shows an example of a point cloud, this point cloud shows a scan of a bridge crossing a river. (Point Cloud Processing / Data Management | H2H Associates, 2018)*

### Point Cloud Library: Point Cloud Data

The application produced in this project requires a way to save and load point clouds, the point cloud data (PCD) will be used as the file format that these point clouds are saved in.

The PCD is a file format which has been developed along side the PCL to aid in the representation of point clouds of any length. The PCD file format has been designed specifically to complement existing file formats that for whatever reason did not or could not support the same functionality that the PCD file format could.

The PCD file format is not the first file format that can support the storage of point cloud data. In the display of computer graphics for games and industry, numerous file formats have already been created to be used in the application of storing a list of arbitrary polygons and point clouds as models and computer aided design (CAD) files.

A selection of file formats which could be used to store point cloud data other than the PCD file format include the following file formats.

The PLY file format. The PLY file format is a file format developed to store polygon meshes, the PLY file format was developed at Stanford University. (*The PLY Polygon File Format*, no date)

The STL file format is another file format that could store point cloud data. The STL file format is a file format developed for use with stereo lithography CAD software, the STL file format was developed by 3D Systems. The STL file format has also become very popular in the 3D printing community. (*STL File Format (3D Printing) - Simply Explained* | *All3DP*, no date)

The OBJ file format is a file format developed to represent the polygons and geometry of meshes for use in 3D graphics, the OBJ file format was initially developed by Wavefront Technologies.

The X3D file format is a file format developed to be the International Organization for Standardization (ISO) standard file format to represent the polygons and geometry of meshes for use in 3D graphics using the extensible markup language (XML).

The file formats mentioned above suffer from a number of disadvantages when being used for the purpose of representing point clouds. This can be understood because these file formats were not designed to represent point clouds specifically, most of these file formats were developed for a different purpose at a time before the depth sensing technology and the algorithms used to manipulate their output had been invented.

The PCD file format was developed specifically for the task of holding point cloud data, and because of this, the PCD file format has the advantage over the file formats mentioned above as it offers a greater degree of flexibility and speed.

The PCD file format also has the ability to store and process either organised or unorganized point cloud datasets, this is of even more importance when the application in question runs calculations on point clouds in real time.

The PCD file format includes an interface for binary data storage, binary data storage is the fastest possible way to both load and save data. (Bayardo *et al.*, 2004)

The PCD file format also includes interfaces that can handle the storage of data using multiple different data types, including all primitive data types such as, characters, short integers, integers, floating point numbers and double precision floating point numbers. The PCD file format can also handle data that is both signed and unsigned.

The PCD file format can handle invalid depth points, these points are stored as a NAN value and can be removed with ease to reduce the amount of memory and processing time consumed by a data set. This is very important in fields such as computer vision where many large data sets may be computed either simultaneously or concurrently.

However, the most significant advantage of the PCD file format is that because the PCD file format is developed concurrently by members of the PCL project, the PCD file format can be custom tailored for the specific requirements of the PCL, thus the highest performance, with respect to the PCL, can be achieved using the PCD file format. This is in contrast to attempting to adapting a different file format to work as the standard file format for the PCL, this would probably incur additional delays through conversion functions.

Each PCD file starts with a header that identifies and declares certain variables to be used with the loading and processing of a given point cloud. The header of a PCD must always be encoded using American standard code for information interchange (ASCII), this is because this header file describes how to load a binary file and without knowing the encoding scheme of a given file, that file could never be loaded itself. The variables contained within the header file as well as any data encoded using ASCII is separated using a Unix style new line.

The version variable of a PCD file specifies the specific version of PCD that a file has been stored as.

The field variable of a PCD file specifies the name of each dimension of a point cloud. For instance, a point cloud may only contain the three dimensions required to represent a point cloud or it may contain these as well as RGB colour information. (*Color supported generalized-ICP*, no date)

The size variable of a PCD file specifies the size of each dimension of a point cloud in bytes. For instance, a signed or unsigned character is one byte, a signed or unsigned short integer is two bytes, a signed or unsigned integer or floating point number is four bytes and a double precision floating point number is eight bytes.

The type variable of a PCD file specifies the variable type of each dimension of a point cloud. For instance, I is a signed character, signed short integer or signed integer, U is a unsigned character, unsigned short integer, or unsigned integer and F is a floating point number or double precision floating point number.

The count variable of a PCD file specifies how many instances of each dimension exists for each point cloud. For instance, point clouds usually only contain one instance of each dimension, however if a histogram was being generated using something like the the viewpoint feature histogram (VFH) this may have up to 308 instances of each dimension. Using this method allows PCL to treat a point cloud as a single contiguous block of memory. If the count variable of the PCD file is not set all dimensions have only one instance by default.

The width variable of a PCD file specifies the horizontal resolution of the point cloud. For instance, if the point cloud is an unordered point cloud it is assumed that the point cloud is a one dimensional object, thus the width variable of the point cloud represents the total number of points in the point cloud and should be equal to the points variable. If the point cloud is an ordered point cloud the width variable represents the number of elements in each row of the point cloud.

The height variable of a PCD file specifies the vertical resolution of the point cloud. For instance, if the point cloud is an unordered point cloud it is assumed that the point cloud is a one dimensional object, thus the height variable of the point cloud is set to one. If the point cloud is an ordered point cloud the height variable represents the number of elements in each column of the point cloud.

The viewpoint variable of a PCD file specifies the position of the origin of the coordinate system used to visualise the point cloud. This variable can be used later when trying to calculate transformations between two different coordinate systems.

The points variable of a PCD file specifies the total number of points in a point cloud. Because the points variable can be calculated from the width variable and the height variable the points variable is mostly deprecated.

The data variable of a PCD file specifies the encoding system which has been used to store the point cloud data type that the point cloud data is stored in. For instance, if the data is set to ASCII then the point cloud data has been encoded using ASCII encoding, if the data is set to binary then the point cloud data has been encoded using binary encoding.

An example of a PCD file can be seen in the appendix bellow. (Appendix L:)

## Point Cloud Library: Cleaning

Cleaning a point cloud involves removing unnecessary points from that point cloud, this is done to improve the accuracy of the registration and tracking algorithms. Cleaning is also performed in order to cut down on the processing time that is taken up by the registration and the tracking algorithms. The raw point cloud is likely to contain many more points than are necessary for the registration algorithm to accurately register point cloud to point cloud. For each unnecessary point in each given point cloud the time of execution of the application increases dramatically, therefore it is advantageous to clean each point cloud. Two standard cleaning methods include de noising and downsampling.

De noising is the process of removing extraneous or outlying points, these points represent noise that has been added to the point cloud from sources such as the error of the depth sensing camera or from reflections off of shiny surfaces. Statistical outlier removal (SOR) is an example of a de noising algorithm, SOR works by iterating through every point in a given point cloud and finding for a selection of points around said initial point the standard deviation of this group of points, if the initial point is outside a given standard deviation of this group of points it is considered to be noise and is removed. (Rusu and Cousins, 2011)

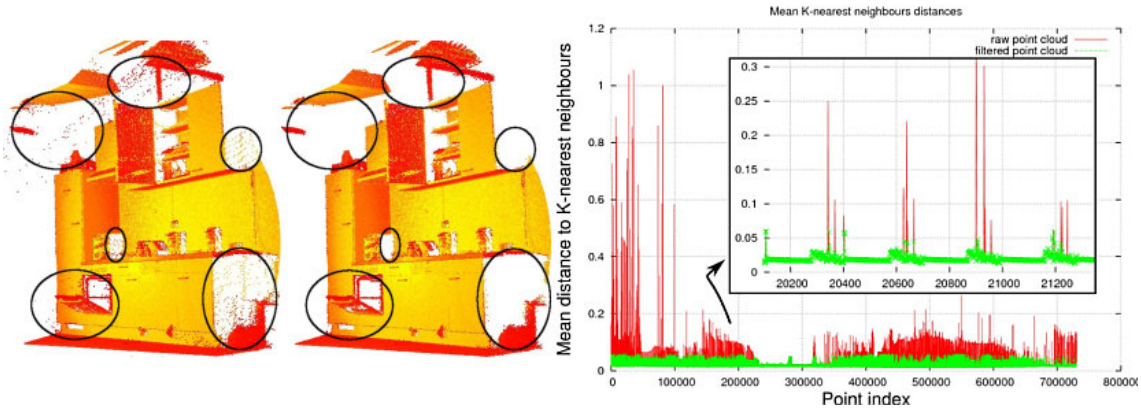


*Illustration 16: This illustration shows an example of the effectiveness of SOR, the image on the left shows a raw point cloud and then the image on the right shows the same point cloud once SOR has been applied. (Documentation - Point Cloud Library (PCL), no date c)*

One method to downsample a point cloud is using a voxel grid filter (VGF). VGF works by passing a filter, of a given voxel size, over a point cloud and then averaging

every point inside the voxel to one individual point. Specifically, this is done by taking a voxel grid and passing it over the point clouds, then, for each voxel, the centre of gravity of the points contained within that voxel is calculated. Thus, then the centre of gravity of the voxel can reflect an average of the points from within that voxel, this centre of gravity replaces the points from within the voxel in a new point cloud.

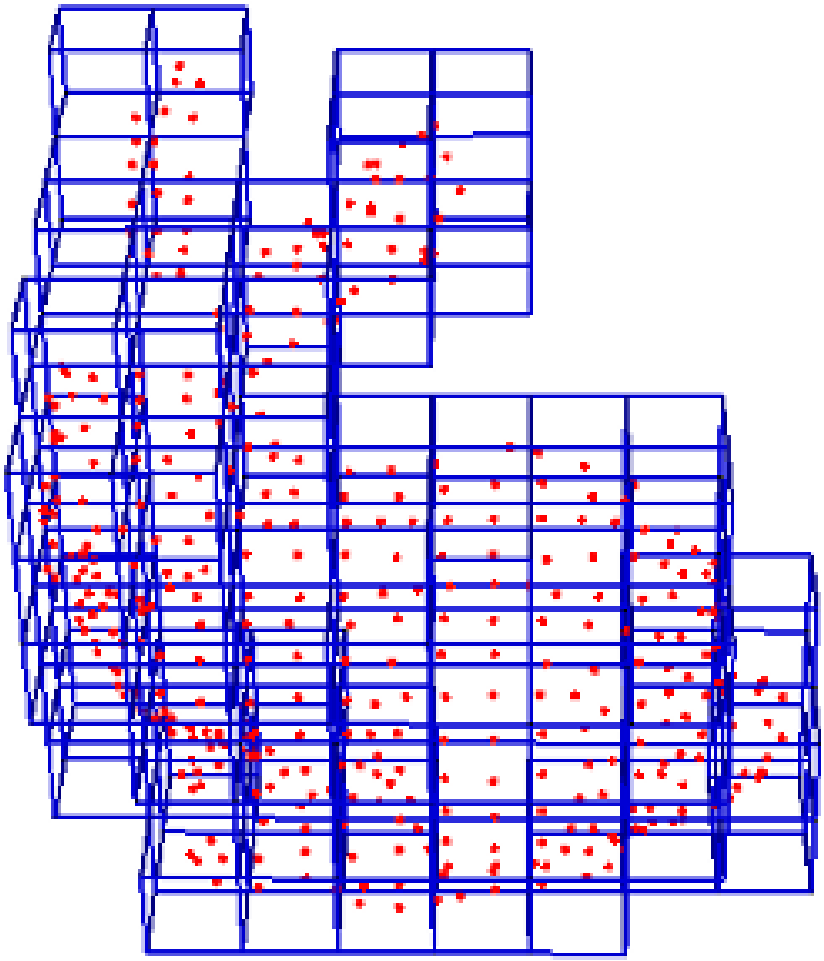

*Illustration 17: This illustration shows an example of the voxels, which are used during VGF, passing over a point cloud. (Documentation - Point Cloud Library (PCL), no date d)*

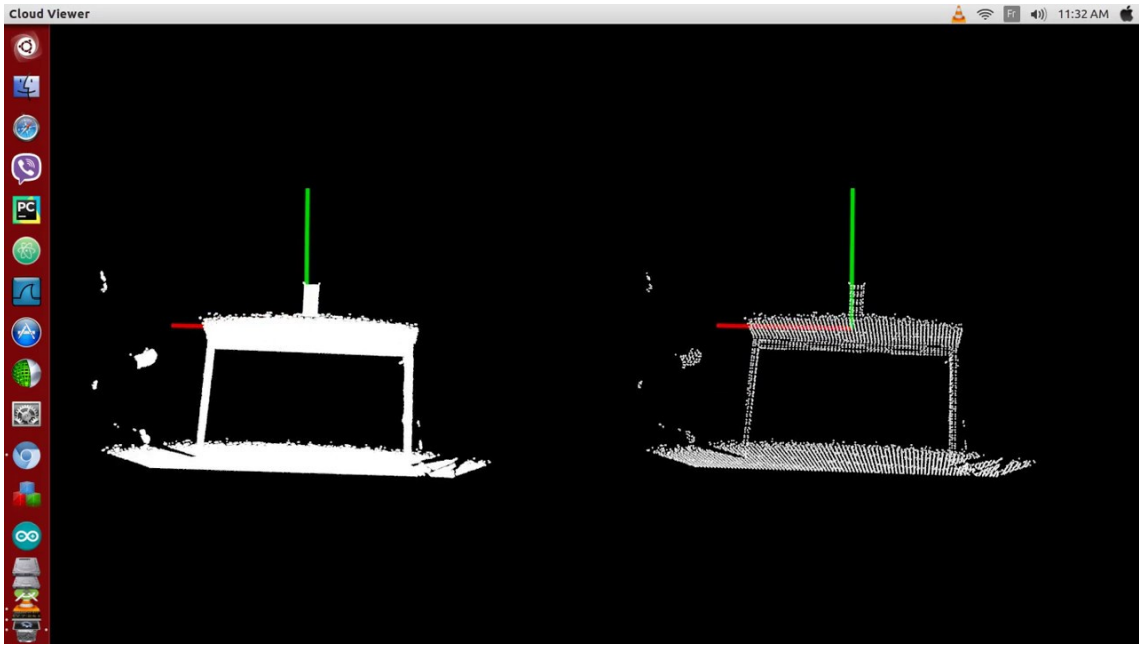


*Illustration 18: This illustration shows an example of the effectiveness of VGF, the image on the left shows a raw point cloud and then the image on the right shows the same point cloud once VGF has been applied. (pcl downsample voxel grid - YouTube, no date)*

The application produced in this project will make use of the cleaning algorithms mentioned above to improve the accuracy and speed of the registration and tracking algorithms.

## Point Cloud Library: Registration

Registration is the process through which two point clouds are aligned into one coherent, complete model. The way in which these two point clouds are aligned is that a transformation is found that causes the two point clouds to, as near as possible, perfectly overlap with one another. Thus for each pair of point clouds a method must be found that can calculate this transformation and then output the transformation between these two point clouds.

The process of registering a pair of point clouds to one another can be referred to as pairwise registration, the output of pairwise registration is usually a homogeneous rigid transformation matrix which represents the scale, skew, rotation and transformation that would have to be applied to one point cloud to align it to the other point cloud.

ICP is an example of a registration algorithm. The ICP registration algorithm finds the rigid transformation between a source ROI point cloud and a target ROI point cloud. The transformation is found by iterating through every point in one point cloud and finding a transformation that will take this point to its closest neighbour in the other point cloud, once a transformation has been found, for every point, these transformations are summed together and averaged and then applied to the point cloud in question. The transformation that is calculated for each iteration is recorded along with the sum of the difference between all the points in the source point cloud to the points in the target point cloud. This process is then repeated in order to minimize the error function calculated as the difference or distance between the two point clouds. (Besl and McKay, 1992)

Specifically the algorithmic steps which must be completed for each iteration of the ICP registration algorithm would be. First, for each point in the first point cloud find the Euclidean closest point in the second point cloud.

Then, calculate a transformation that is a combination of rotations and translations in the horizontal, vertical and depth planes using a technique that aims to minimise the root mean square deviation (RMSD) or root mean squared error (RMSE) of the points in question. The RMSD or RMSE is calculated as the square root of the differences between the positions of the two points in question, this could also be referred to as the quadratic mean of the differences between the positions of the two points in question. (Levinson, 1946)

This RMSD or RMSE is then used to determine the transformation which will best align each point in the first point cloud to its neighbour found in the second point cloud. During this step points may be rejected or otherwise removed, prior to alignment, from both point clouds if they are detected as having an excessive distance to their nearest neighbour, in this case they are deemed to be outliers.

Finally, the first point cloud is transformed using the transformation found in the previous step before the process is started again from the first step. Registration ends when the error between the two point clouds is calculated as being under a certain acceptable amount or if a certain number of iterations has already occurred.

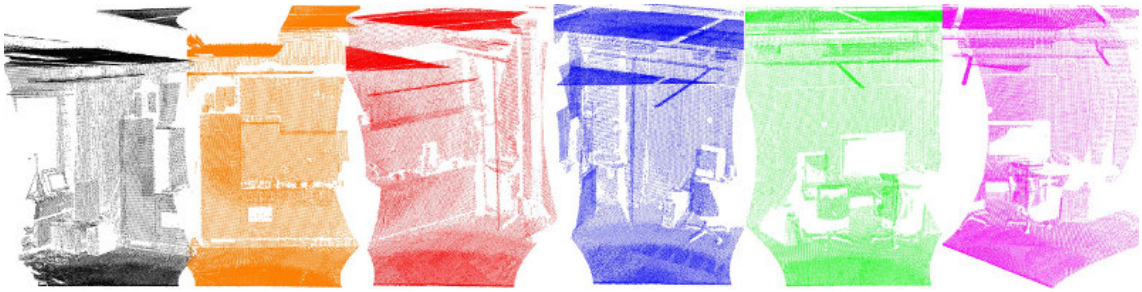

*Illustration 19: This illustration shows an example of the effectiveness of ICP, this illustration shows a collection of raw point clouds.(Documentation - Point Cloud Library (PCL), no date b)*

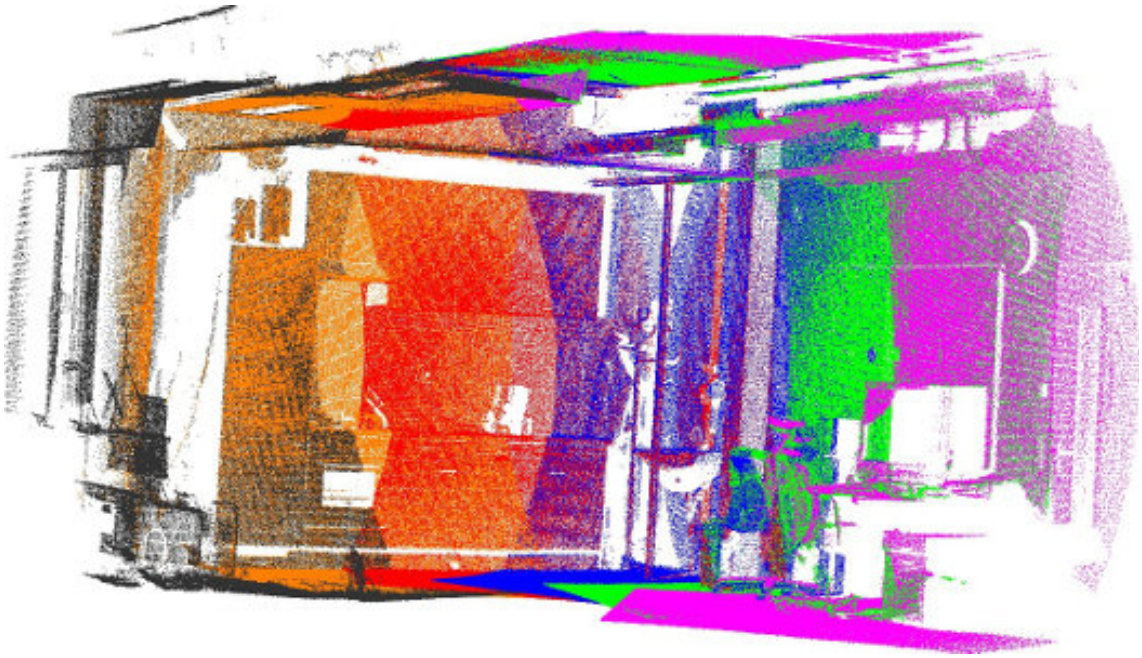

*Illustration 20: This illustration shows an example of the effectiveness of ICP, this illustration shows the same point clouds as above (Illustration 19) once ICP has been applied. (Documentation - Point Cloud Library (PCL), no date b)*

The normal distributions transform (NDT) is another example of a registration algorithm. NDT is more efficient that ICP when working with large data sets, this is because NDT attempts to estimate what the next most likely transformation is between two point clouds by using a set of standard optimisation algorithms which are applied to a statistical model generated from the two point clouds themselves. (Magnusson, 2009)

The application produced in this project will make use of the transformation output from the registration process as one of the ways in which the difference between two point clouds and hence the motion contained in these point clouds is calculated.

The application produced in this project will initially make use of the ICP registration algorithm, thus it is important to understand the way in which this algorithm works in order to be able to produce a solution which is of the highest quality possible. However, the application will be written in modular fashion so that the NDT registration algorithm, or any other kind of registration algorithm, could be added in the future with as few compatibility issues as possible.

It is important to note however that both the ICP and NDT registration algorithms produce a very similar output to one another, the only major difference between the two algorithms is the speed and efficiency with which they come to their conclusion.

### Point Cloud Library: Tracking

Registration can also be extended to offer tracking functionality. Tracking is the process through which the position of a specific model is located within a given scene over time. The difference between the positions of this model in each given frame of an acquisition can be used to extrapolate the motion, velocity and acceleration of a given object.

Registration can be used as a tracking metric by using the same source point cloud for each iteration of the registration process. This source point cloud should contain a model of the object that is to be tracked in a given scene. For instance, if the head of a subject was to be tracked over an acquisition, then a model of the head of the subject would have to be segmented from the subject as a whole. If this model was then used as the source point cloud during the registration process, then the registration algorithm would align this model to the head of the subject in every subsequent point cloud. Thus the registration algorithm would calculate a transformation to the head of the subject every time it was run.

*Illustration 21: This illustration shows an example of the effectiveness of registration as a tracking algorithm, this illustration shows a blue collection of points tracking a red collection of points. (Documentation - Point Cloud Library (PCL), no date a)*

The application produced in this project will make use of this pseudo tracking algorithm to attempt to add the ability for the application to extract a surrogate or breathing signal from a given set of point clouds. This surrogate or breathing signal could then be used as discussed in the alternative solution using data driven motion tracking. (Bousse *et al.*, 2016) (Bousse *et al.*, 2017)

This would be specifically facilitated by segmenting a section of the subjects chest into a separate point cloud and then attempting to align this point cloud with every subsequent frame from the acquisition. This should return the position of the same section of the subjects chest after each registration iteration, the distance between this section of the subjects chest over time can be used to estimate the target surrogate or breathing signal.

## Software For Tomographic Image Reconstruction: Overview

STIR (*UCL/STIR: Software for Tomographic Image Reconstruction*, 2018) is an open source framework for the tomographic image reconstruction of data from PET/CT scanners currently developed at University College London by the Institute of Nuclear Medicine.

The aim of STIR is to provide a cross platform, object oriented framework for many different kinds of data manipulation in tomographic imaging. STIR focuses on the iterative image reconstruction of data from PET/CT, PET/MR and SPECT scanners, however, in the future, other areas of research applications and different types of imaging modalities will be added. STIR is written using the C++ programming language but interfaces do exist for both the Python and MATLAB scripting languages.

STIR has been in production since 2000 and has been developed at several institutions and companies. STIR had been used in many applications and by many researchers from all over the world. STIR has been cited in over 200 publications. (Thielemans *et al.*, 2012)

The application produced in this project will be designed in such a way as to be as compliant with the existing programming standards used in STIR as possible, this is because the application produces in this project may have elements that could be integrated into STIR.

## Sinogram Acquisition Viewer: Overview

SAvVy (*NikEfth/SAvVy: SAvVy - Stir dAta Viewer*, 2018) (*NikEfth/PPDD: SAvVy - Stir dAta Viewer*, no date) is an open source application for opening, manipulating and viewing sinograms and other medical image files developed at the University of Hull by the PET preclinical centre.

SAvVy is written using the C++ programming language and uses a GUI which was developed using the Qt GUI toolkit. SAvVy uses the STIR framework as its back end for image reconstruction.

The application produced in this project will be designed in such a way as to be as compliant with the existing programming standards used in SAvVy as possible, this is because the application produces in this project may have elements that could be integrated into SAvVy. This is because SAvVy has the capacity to load other applications which are written using the C++ programming language and the Qt GUI toolkit, the application produced in this project will be designed so that it can be loaded, as is, by SAvVy and can then be used to provide motion correction facilities to the PET/CT data loaded by SAvVy.

**Language: Overview**

Because the vast majority of the libraries, which have been selected to aid in the development of the application, are written using either the C or C++ programming language the application, which is to be produced in this project, will also be written using the C++ programming language so as to provide the greatest amount of synergy between itself and the libraries that it intends to make use of.

An advantage of using the C++ programming language is that it is an unmanaged language. This means that the developer of an application has to manage the creation and deletion of objects in memory. In the C++ programming language there is no garbage collector, like there may be in managed languages, which is always running in order to clean up the memory that the application makes use of. The garbage collector, as well as a host of other small design choices, usually cause managed languages to perform much slower than unmanaged languages. As such the application which is to be produced in this project will be written using the C++ programming language in order to reap the speed benefits of using an unmanaged language. (Ray *et al.*, 2014)

A disadvantage of using the C++ programming language is that it usually takes a great deal longer to implement any given aspect of an application using this language. This is because the C++ programming language does not provide many native features and as such it usually takes many more lines of C++ than it would of other programming language, with more native features, to implement the same functionality.

# 4 Technical Development and Achievements

## 4.1 System Design and Implementation

### System Design and Implementation: Overview

The following class descriptions all contain the following unless otherwise stated.

The class contains a default constructor, a destructor which calls the destructor method and copy and move constructors and assignment operators.

The destructor method is a private method which takes in a boolean. If the boolean is true the class is destroyed, if the boolean is false the objects are returned to their initial state.

The class contains an accessor and a mutator to each one of its member variables, this allows for either the value of or a handle to the private variables to be accessed from outside of this class.

The class kill method takes in a boolean and is used to remotely call the destructor method.

### Kinect Application: Overview

The Kinect application was mainly designed to fill the role of connecting to the Kinect camera, retrieving depth images from the Kinect camera and then outputting these depth images in some format.

To perform this role the Kinect application was linked to the OpenKinect Libfreenect and OpenKinect Libfreenect2 libraries in order to access the Kinect camera in a cross platform way. The Kinect application was also linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon these libraries.

The Kinect application can also be used to retrieve RGB images from the Kinect camera and to adjust the motor which controls the tilt of the Kinect camera either up and down by a fixed amount or by an amount input by the user.

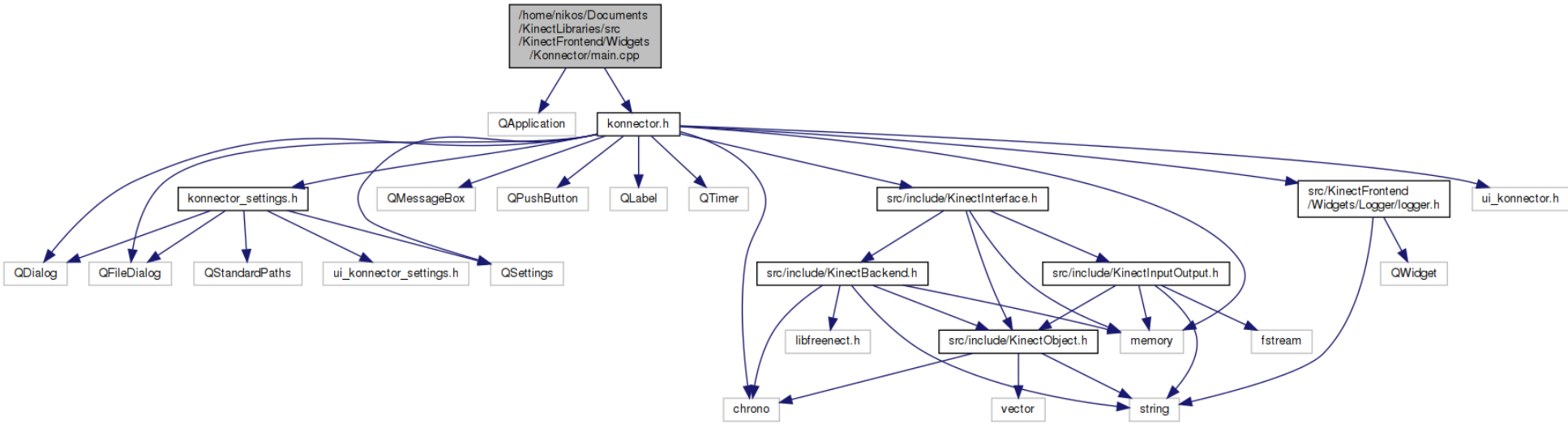


*Illustration 22: This illustration shows the unified modelling language (UML) class diagram for the entire Kinect application as a whole, this diagram shows the interaction between the classes of the Kinect application.*

The Kinect application contains seven classes that are of interest. These classes are the Kinect back end class, the Kinect input output class, the Kinect interface class, the Kinect object class, the Konnector logger class, the Konnector class and the Konnector settings class.

Most of the member variables contained within the classes of this application and some of their methods or functions are private. To be made private means to limit access to these variables and methods to anything from outside of the class that contains them.

The member variables and methods or functions of these classes have been made private in an attempt to follow the encapsulation method of programming, this means that the value or state of important structured data contained within the class is hidden from the user, or other applications, when they are outside of the class that encapsulates them. This is useful as it blocks unauthorised access to sensitive information held within these classes.

Encapsulation can also aid by removing complex helper functions from access outside of the class that uses it. This is useful as these helper functions have no use outside of the class in question so by being accessible they would only allow for the class in question to be broken or otherwise tampered with while only serving to clutter up the GUI. (Micallef, 1987)

## Kinect Application: Activity Diagram

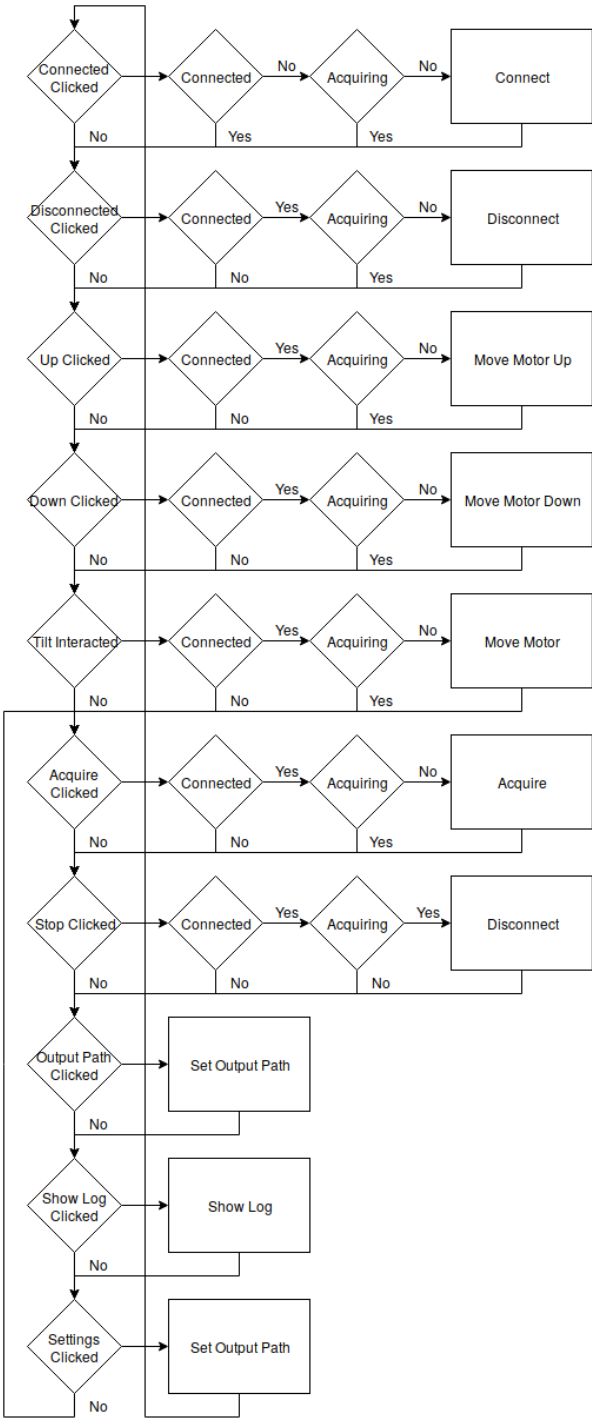


*Illustration 23: This illustration shows the logical flow through the Kinect application as a UML activity diagram.*

### Kinect Application: Kinect Back End Class

The Kinect back end class, this class finds and connects to a Kinect camera. In case of multiple connected cameras, only the first will currently be considered. What should be output can be selected from the frontend.

The Kinect back end class is associated with handling the direct connection to the Kinect camera. This class can connect to a Kinect camera and retrieve depth images from the Kinect camera using callbacks.

To perform this role the Kinect back end class was linked to the OpenKinect Libfreenect and OpenKinect Libfreenect2 libraries in order to access the Kinect camera in a cross platform way. Hence this application depends upon these libraries.

The Kinect back end can also be used to retrieve RGB images from the Kinect camera and to adjust the motor which controls the tilt of the Kinect camera.

The Kinect back end class is a singleton class, this means that the class cannot be instantiated and can only be accessed through an accessor method. The Kinect back end class has been implemented in this way to disallow multiple concurrent connections to the callbacks located within this class.

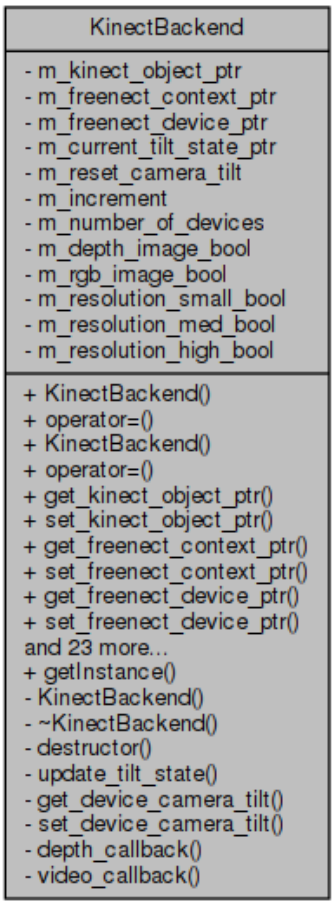


*Illustration 24: This illustration shows the UML class diagram for the Kinect back end class.*

The Kinect back end class contains a default constructor, a destructor which calls the destructor method and copy and move constructors and assignment operators. However, all of these methods are either private or deleted because this class is a singleton.

The depth callback method is a private static method which takes in the metadata of the Kinect camera and returns the current depth image and a timestamp to when the depth image was received.

The get device camera tilt method is a private inline method which retrieves the current camera tilt from the Kinect camera.

The get instance method is a static method which returns a reference to the singleton Kinect back end class.

The Kinect back end kill method takes in a boolean and is used to remotely call the destructor method. In this instance it is used specifically to disconnect from the Kinect camera.

The Kinect back end main method is used to connect to a Kinect camera.

The update method is used to check the output from the callbacks from the Kinect camera, if the callbacks have new data the method populates the buffers in memory with the new data and sets flags so that other parts of the application know that new data has been acquired.

The update tilt state method is a method which is used to update a variable to signal the state of the tilt motor of the Kinect camera, these states are whether the Kinect camera is moving, stationary or at the limits of its movement.

The video callback method is a private static method which takes in the metadata of the Kinect camera and returns the current RGB image and a timestamp to when the RGB image was received.

The current tilt state pointer variable holds a pointer to the object that represents the current state of the tilt motor of the Kinect camera.

The Libfreenect context pointer variable holds a pointer to the object that holds metadata about the possible connections of the computer on which the application is running.

The Libfreenect device pointer holds a pointer to the object that holds metadata about the currently connected Kinect camera.

The increment variable holds the step size that the Kinect camera tilt motor will be adjusted by.

The Kinect object pointer variable holds a shared pointer to the object that represents the output from the Kinect camera.

The number of devices variable holds the total number of Kinect cameras currently connected to the application.

The reset camera tilt variable holds the angle that the Kinect camera should be reset to.

The Kinect back end class contains variables which are set in the settings section of the application and serve only to update the application based on the users input. These include variables which reflect the resolution that should be used for the current acquisition and variables which reflect if depth and RGB images should be output from the program.

## Kinect Application: Depth Image Pseudo Code

```
Acquisition speed equals acquisition speed from settings

Acquire depth image equals acquire depth image from settings

Output text depth image equals output text depth image from settings

Output path equals output path from settings

Output binary depth image equals output binary depth image from settings

Connect to Kinect camera

Start acquisition timer

While Kinect camera is connected

    If acquisition timer is greater than acquisition speed

        If acquire depth image

            If new depth image has been acquired from Kinect camera

                For horizontal resolution from Kinect camera multiplied by
                vertical resolution from Kinect camera

                    Data at index equals data from Kinect camera at
                    index

                New data acquired equals true

        Acquisition timer equals zero

    If new data equals true

        If output text depth image

            Output data using text encoding at output path

        if output binary depth

            Output data using binary encoding at output path

            Output header file which points to output
```

The above pseudo code shows the process by which a depth image is acquired and output from the Kinect back end class. This only shows one thread of logic and it assumes that connecting to and acquiring data from the Kinect camera callbacks is trivial.

First, the relevant variables to control the flow of logic through the code are set from the variables in the application settings.

Then, a connection to the Kinect camera is established before starting the acquisition timer. The acquisition time is used to trigger an event when the acquisition timer elapses, which in this case is represented by the while loop and the if statement.

If an acquisition from the Kinect camera has been made in the time since the last update step and a depth image is to be acquired then the contents of the depth image buffer are copied into the data buffer. The process is exactly the same for a depth image as it is for an RGB image.

Finally, the data buffer is written to an output file depending upon the settings which were established at the beginning of execution. If the data buffer is output using binary encoding then a Kinect point cloud library processing (KPCLP) header file is also produced so that this data can be loaded into the point cloud application later.

### Kinect Application: Kinect Input Output Class

The Kinect Input Output class. This class outputs the current data in the Kinect object class, this class outputs every time that the Kinect back end class receives a new frame. The exact form of the output can be selected from the Konnector class frontend.

The Kinect input output class is associated with handling the output of the depth images to file.

The Kinect input output class can also be used to handle the output of internal logs into buffers which can be accessed elsewhere in the application.

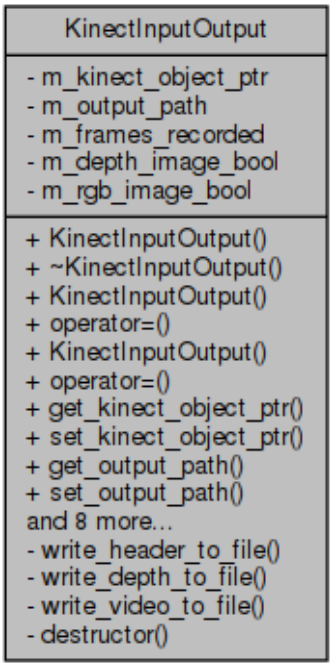


*Illustration 25: This illustration shows the UML class diagram for the Kinect input output class.*

The write depth to file method is a private method which writes the contents of the depth buffer to file using either text or binary encoding.

The write header to file is a private method which takes in the metadata relating to the acquisition and writes it to file using text encoding.

The write video to file method is a private method which writes the contents of the video buffer to file using either text or binary encoding.

The Kinect input output class contains variables which are set in the settings section of the application and serve only to update the application based on the users input. These include variables which reflect if depth and RGB images should be output from the program and the path to where the depth and RGB images should be output to.

The frames recorded variable holds the total number of frames output per acquisition.

The Kinect object pointer variable holds a shared pointer to the object the represents the output from the Kinect camera.

### Kinect Application: Kinect Interface Class

The Kinect Interface class. This class holds pointers and or references to the back end functions of the Kinect library. The frontend classes can access the back end classes through this interface.

The Kinect interface class is associated with abstracting the Kinect back end class, the Kinect input output class and the Kinect object class from the Konnector class. The Kinect interface class contains pointers and references to these classes and if the Konnector requires access to an element of any of these classes it must go through the Kinect interface class. This has been implemented to attempt to abstract the frontend from the back end of the application.

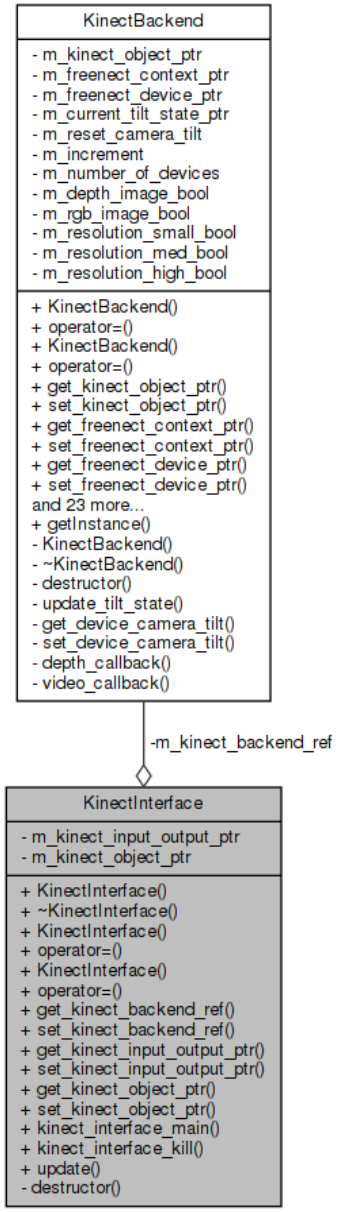


*Illustration 26: This illustration shows the UML class diagram for the Kinect interface class.*

The Kinect input output main method is used to create and store an instance of the Kinect back end class, the Kinect input output class and the Kinect object class.

The update method calls the update methods of the Kinect back end class, the Kinect input output class and the Kinect object class.

The Kinect back end pointer variable holds a shared pointer to the object that deals with the back end of the application.

The Kinect input output pointer variable holds a shared pointer to the object that deals with the input and output of the application.

The Kinect object pointer variable holds a shared pointer to the object the represents the output from the Kinect camera.

### Kinect Application: Kinect Object Class

The Kinect Object class. This class holds the data received from the Kinect camera, this class also holds metadata regarding the connection to the Kinect camera and the resolution of its output etc.

The Kinect object class is associated with holding the data of the connected Kinect camera. This class not only holds the object which represents the connected Kinect camera but also holds buffers which are used to hold the most recently acquired frames from the Kinect camera, it also holds variables which represent the current state of the Kinect camera.

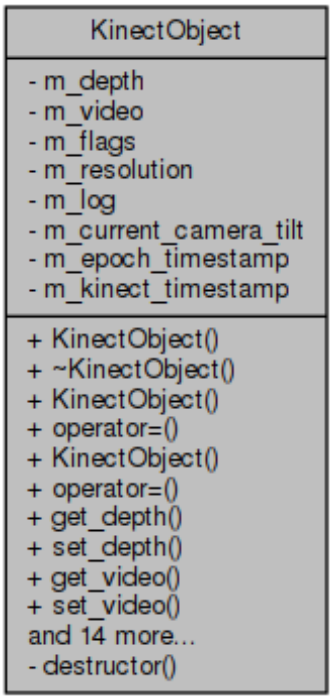


*Illustration 27: This illustration shows the UML class diagram for the Kinect object class.*

The current camera tilt variable holds the angle that the Kinect camera is currently tilted to.

The depth variable is an array or linked list that holds the most recent depth image that has been received from the Kinect camera.

The epoch timestamp variable holds an object that represents the time since epoch (the time since 1st of January 1970 in seconds also known as portable operating system interface (POSIX) time) that the most recent acquisition occurred.

The flags variable is an array or linked list that holds flags representing if new depth or RGB image data has been received from the Kinect camera.

The Kinect timestamp variable holds the time since the most recent acquisition occurred.

The log variable holds a log of events that have happened during the acquisition.

The resolution variable is an array or linked list that holds the resolution of each depth or RGB image.

The video variable is an array or linked list that holds the most recent RGB image that has been received from the Kinect camera.

### Kinect Application: Konnector logger Class

The Konnector logger class. This class is a Qt widget that is to be used in the Konnector frontend class. This class can be used to output logs to the screen for the user to read.

The Konnector logger class is associated with writing any output from the back end of the application to the screen.

To perform this role the Konnector logger class was linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon this library.

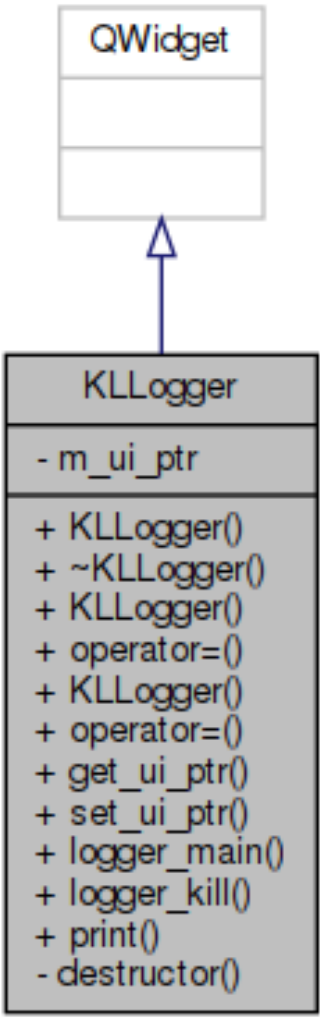


*Illustration 28: This illustration shows the UML class diagram for the Konnector logger class.*

The print method takes in a string of text and prints it to the screen.

The GUI pointer variable holds a pointer to the object that represents the Qt GUI widget of this class. This variable indicates that this is a Qt object.

### Kinect Application: Konnector Class

The Konnector class. This class is a Qt frontend for the Kinect back end classes. This class calls the functions of the Kinect back end classes, collects the output from these classes and then displays this output to the user. This class is also capable of interfacing with the motor of the Kinect camera.

The Konnector class is associated with providing a GUI solution to the user in order to allow them to interact with the application.

To perform this role the Konnector class was linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon this library.

This class provides event handlers that call functions to connect and disconnect from a Kinect camera, adjust the motor which controls the tilt of the Kinect camera either up and down by a fixed amount or by an amount input by the user, start or stop an acquisition and event handlers that call functions to the other GUI elements of this solution.

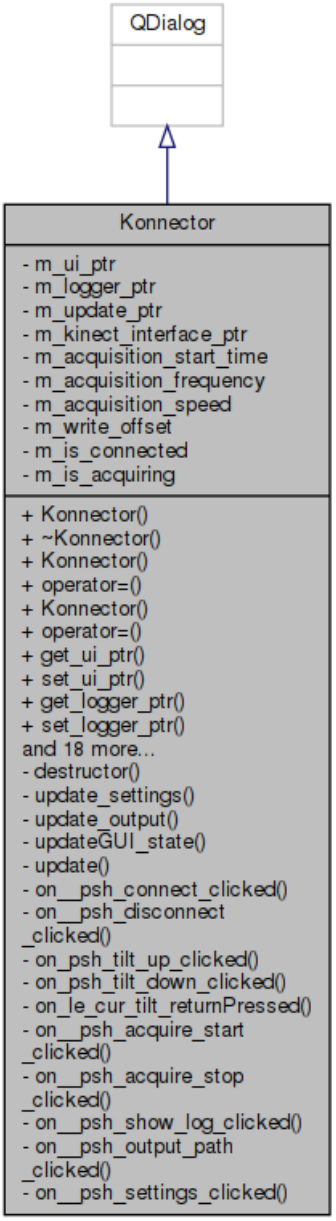


*Illustration 29: This illustration shows the UML class diagram for the Konnector class.*

The acquisition status changed is a signal that causes an event whenever an acquisition is started or stopped.

The camera angle changed is a signal that causes an event whenever the tilt angle of the Kinect camera is changed.

The connection status changed is a signal that causes an event whenever a Kinect camera is connected or disconnected.

The Konnector main method is used to initialise the Qt GUI.

The acquire start method is a private slot method which reacts to the event of the acquire start button being pressed. This method starts an acquisition.

The acquire stop method is a private slot method which reacts to the event of the acquire stop button being pressed. This method stops an acquisition.

The connect method is a private slot method which reacts to the event of the connect button being pressed. This method connects to a Kinect camera.

The disconnect method is a private slot method which reacts to the event of the disconnect button being pressed. This method disconnects from a Kinect camera.

The output path method is a private slot method which reacts to the event of the output path button being pressed. This method changes the output path.

The settings method is a private slot method which reacts to the event of the settings being pressed. This method changes the settings of the application.

The show log method is a private slot method which reacts to the event of the show log button being pressed. This method shows the log.

The current tilt method is a private slot method which reacts to the event of the return key being pressed while inputting a tilt angle. This method sets the current tilt angle to the number in the current tilt box.

The tilt down method is a private slot method which reacts to the event of the tilt down button being pressed. This method sets the current tilt to the number below the current tilt.

The tilt up method is a private slot method which reacts to the event of the tilt up button being pressed. This method sets the current tilt to the number above the current tilt.

The update method is a private slot method which reacts to a timer elapsing at a periodic rate. This method calls the update method of the Kinect interface class.

The update output method is a private method which updates the GUI.

The update settings method is a private method which updates the settings of the application.

The update GUI state is a private slot method which reacts to the signals mentioned above. This method tracks the current state of the GUI.

The acquisition frequency variable holds the frequency at which acquisitions are to take place.

The acquisition speed variable holds the number of milliseconds that need to elapse before another acquisition can take place.

The acquisition start time variable holds an object that represents the time at which the current acquisition started.

The is acquiring variable is true if an acquisition is currently taking place.

The is connected variable is true if a Kinect camera is currently connected.

The Kinect interface pointer variable holds a shared pointer to the object that connects the frontend to the back end.

The Konnector logger pointer variable holds a pointer to the GUI object that writes text to the screen.

The GUI pointer variable holds a pointer to the object that represents the Qt GUI widget of this class. This variable indicates that this is a Qt object.

The update pointer variable holds a pointer to the object that times the duration between updates.

The write offset variable holds the amount of time that must elapse before the GUI can be updated.

**Kinect Application: Use Case Diagram**

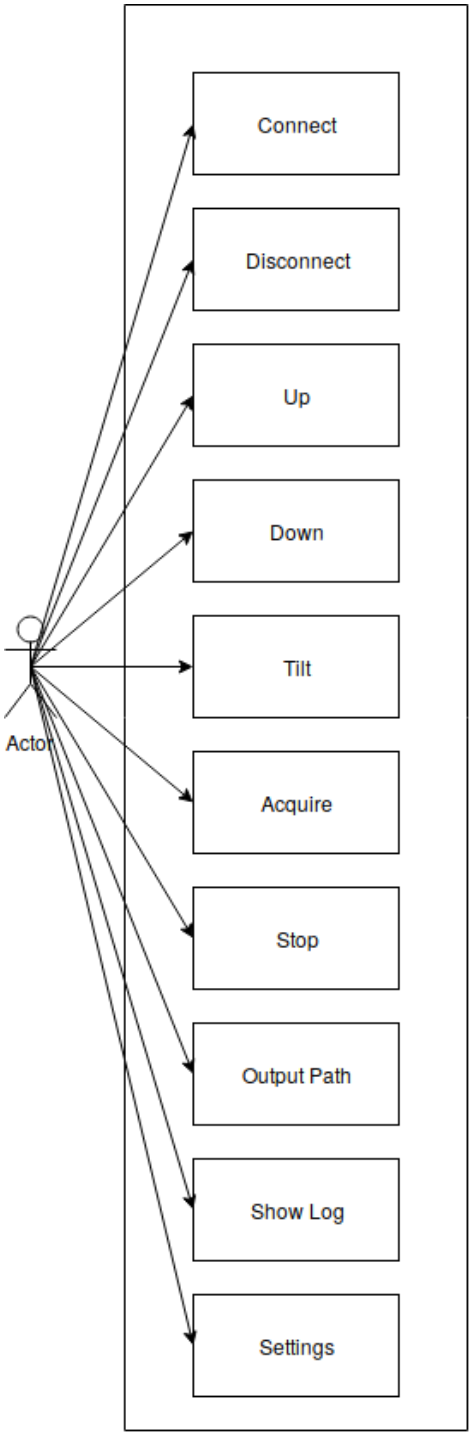


*Illustration 30: This illustration shows the options that the user has in the Kinect application as a UML use case diagram.*

**Kinect Application: Graphical User Interface**

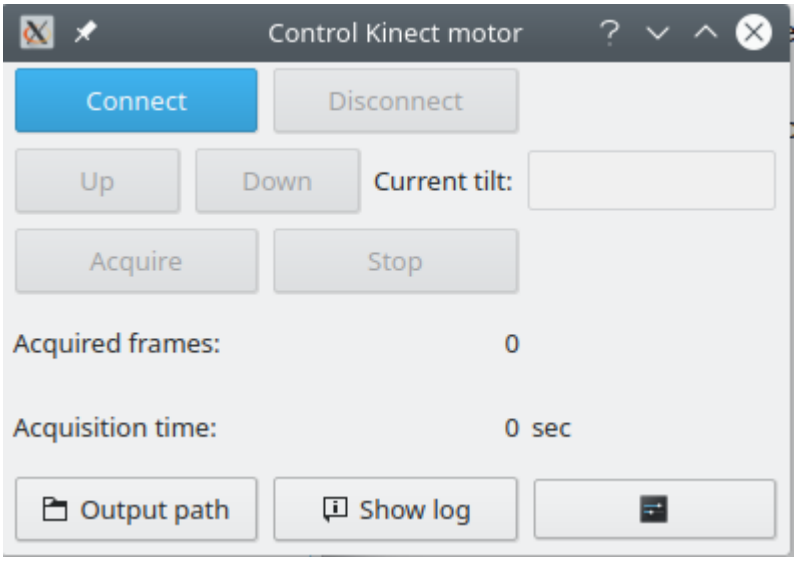


*Illustration 31: This illustration shows the GUI of the Kinect application as it appears when the application is initially started. All options are blocked except for connecting to a Kinect camera and adjusting the settings and opening other Qt widgets.*

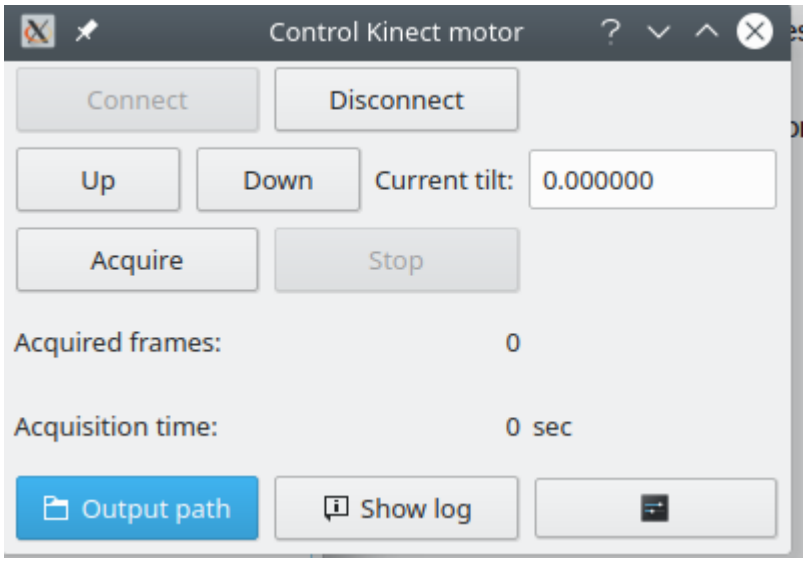


*Illustration 32: This illustration shows the GUI of the Kinect application as it appears after connecting to a Kinect camera. The disconnect, motor up, down and box and acquire options are now available to be interacted with. The connect and stop options are blocked.*

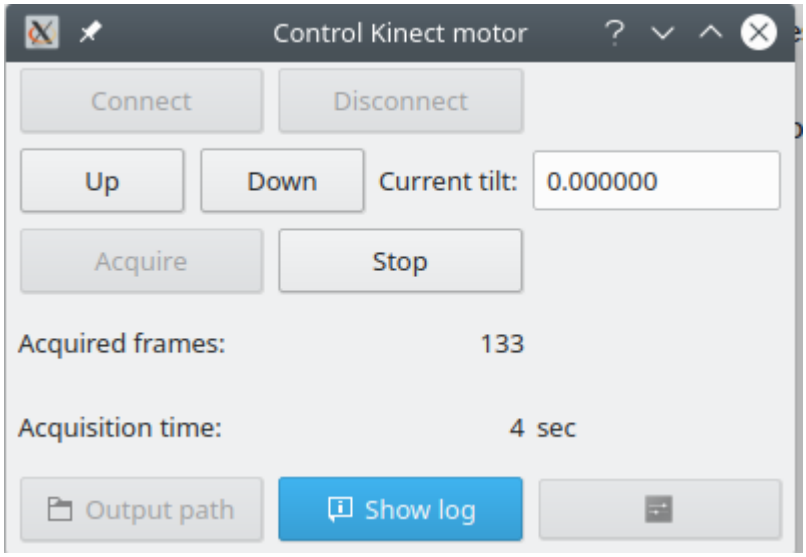


*Illustration 33: This illustration shows the GUI of the Kinect application as it appears after an acquisition has been started. The motor up, down and box and stop options are now available to be interacted with. The connect, disconnect and acquire options are blocked. The options to adjust the settings are also blocked.*

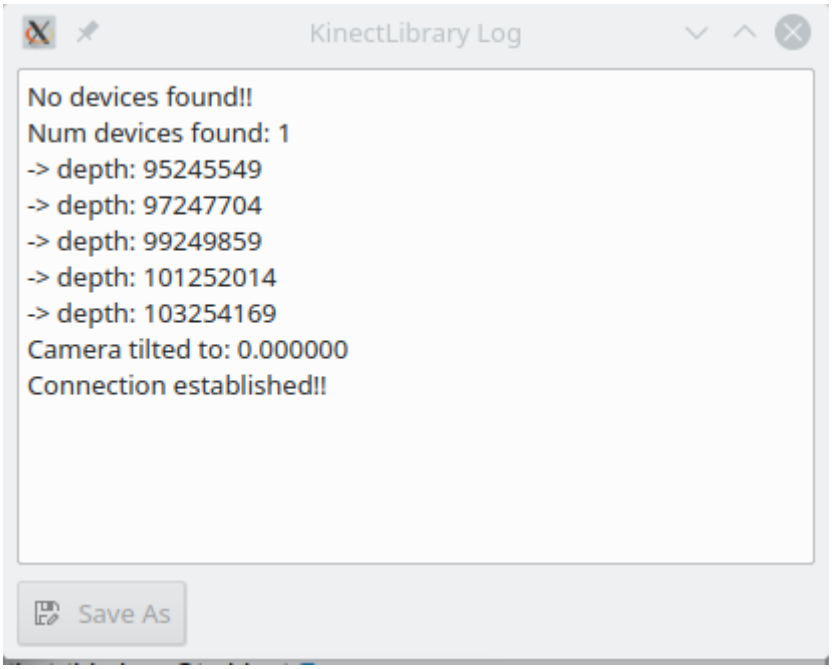


*Illustration 34: This illustration shows the GUI of the Kinect logger application as it appears after connecting to a Kinect camera. Arrows are used to show input and output from the application.*

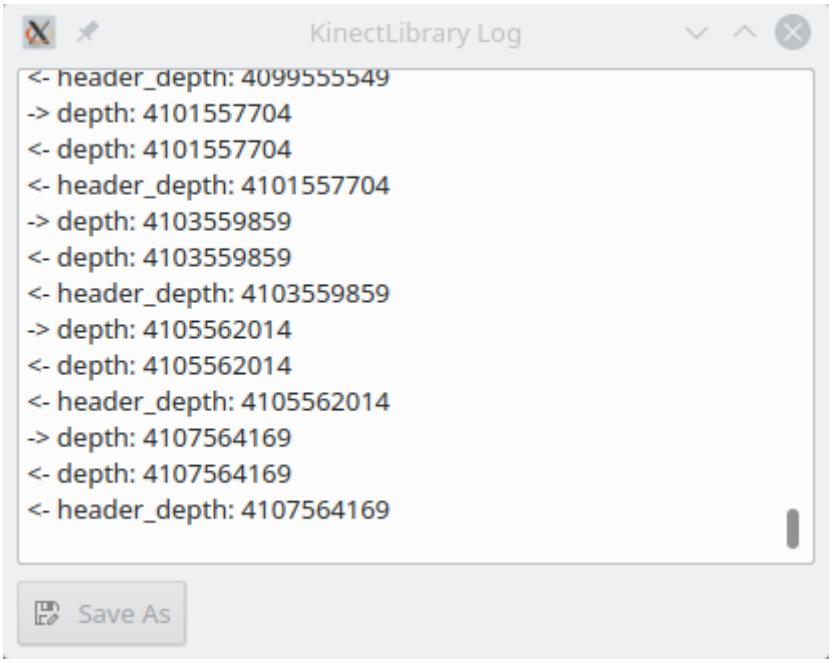


*Illustration 35: This illustration shows the GUI of the Kinect logger application as it appears after an acquisition has been started. Arrows are used to show input and output from the application.*

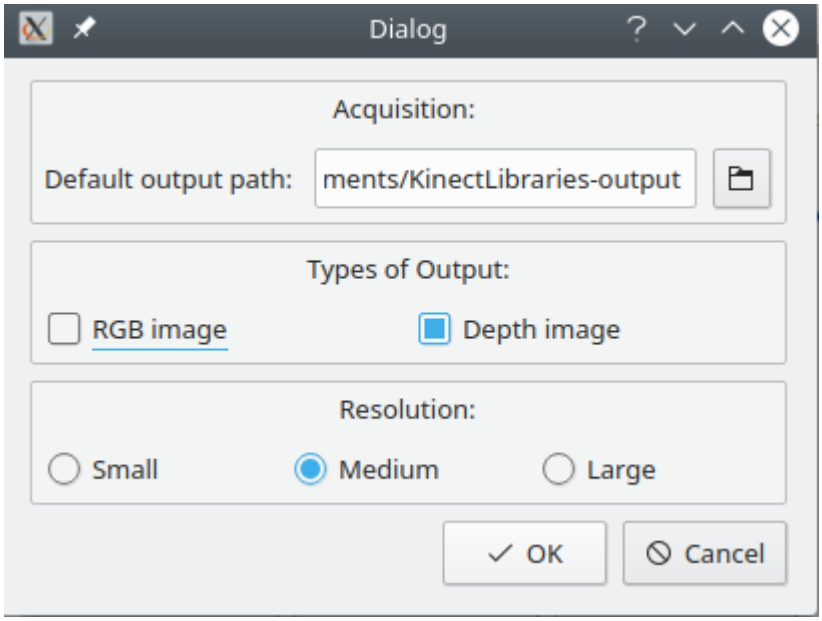


*Illustration 36: This illustration shows the GUI of the Kinect settings application.*

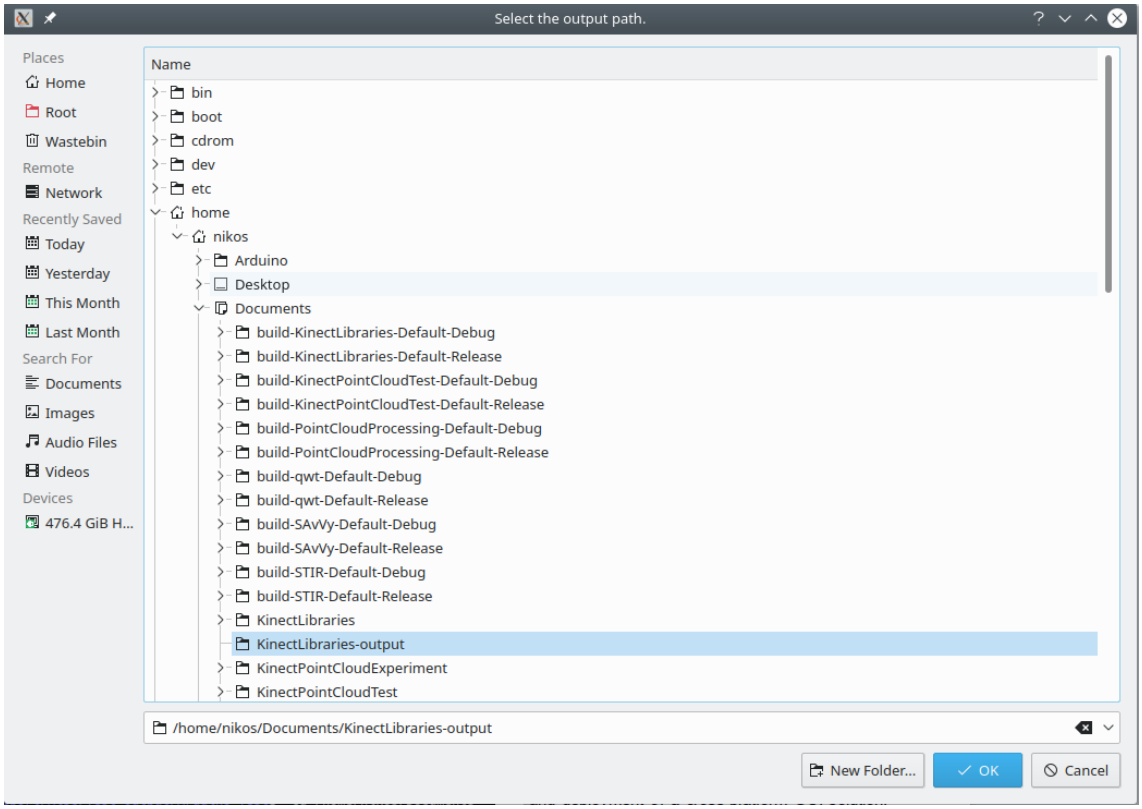


*Illustration 37: This illustration shows the Qt widget used to select paths.*

*Illustration 38: This illustration shows an RGB image visualisation.*

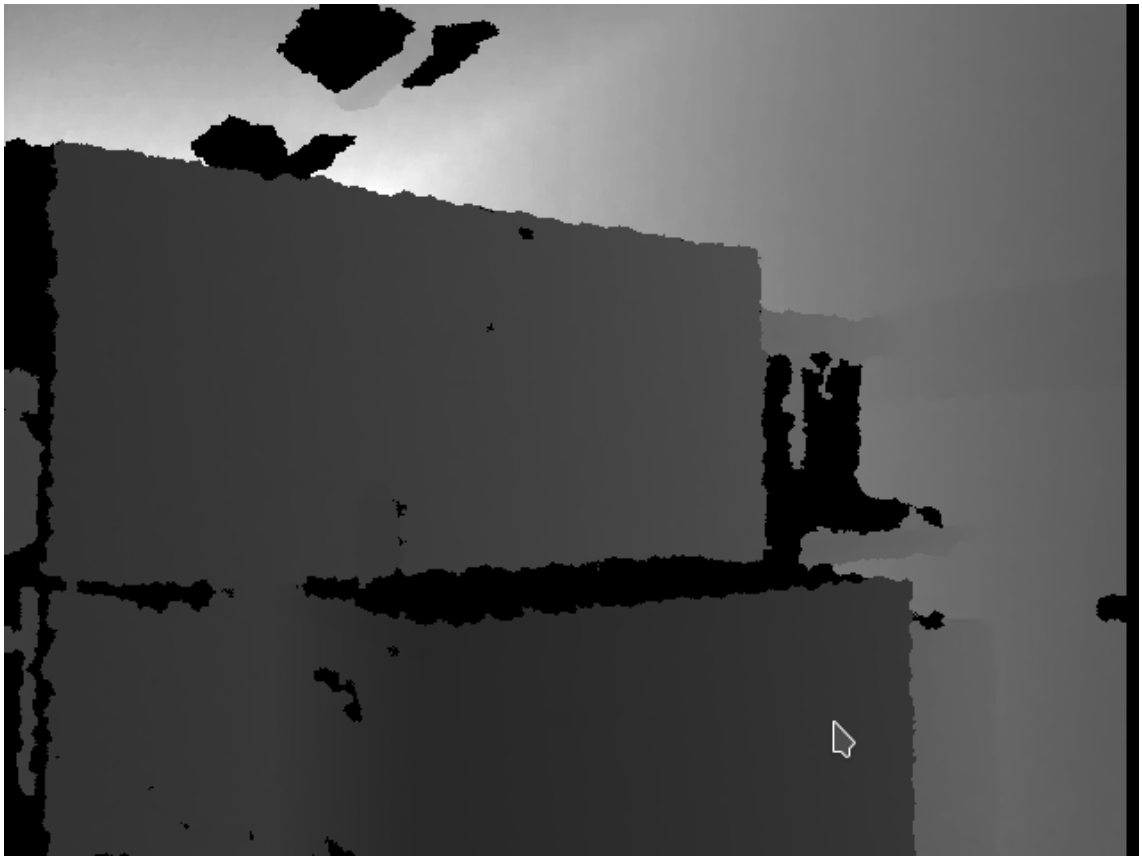

*Illustration 39: This illustration shows a black and white depth image visualisation.*

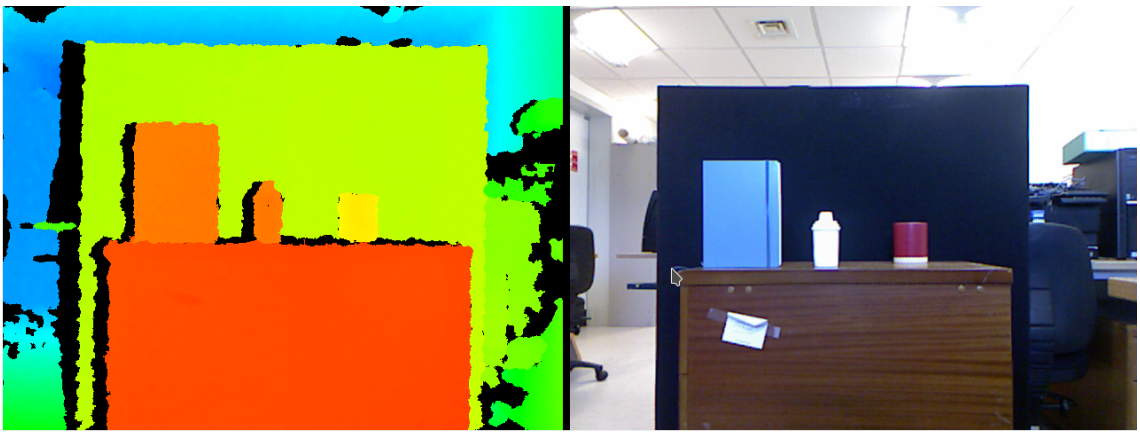

*Illustration 40: This illustration shows an RGB depth image visualisation and an RGB image visualisation of the same scene.*

*Illustration 41: This illustration shows an RGB depth image visualisation.*

### Kinect Application: Konnector Settings Class

The Konnector Settings class. This class holds the settings that are to be used in the Konnector frontend class.

The Konnector settings class is associated with providing a GUI solution to the user in order to allow them the adjust the settings of the application.

To perform this role the Konnector settings class was linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon this library.

This class provides variables which can store the output path of the application and the things to be output from the application.

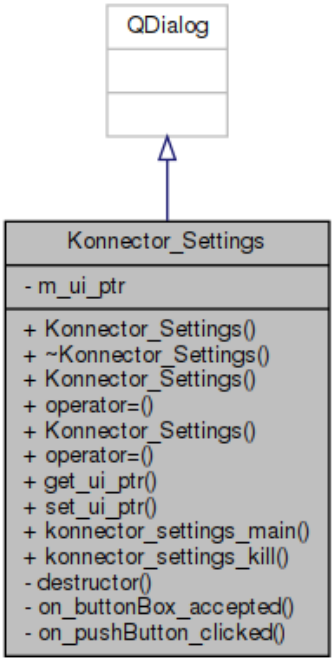


*Illustration 42: This illustration shows the UML class diagram for the Konnector settings class.*

The button method is a private slot method which reacts to the event of the button being pressed. This method updates the saved settings.

The GUI pointer variable holds a pointer to the object that represents the Qt GUI widget of this class. This variable indicates that this is a Qt object.

### Kinect Application: Output

The KPCLP header file holds all of the information necessary to reload the depth images which have been acquired from the Kinect camera.

The KPCLP header version variable represents the version of the KPCLP standard that has been used to encode the information in the KPCLP header file. Currently the most recent version is 0.1.

The data type variable represents the data type that has been used for the acquisition from the Kinect camera. Currently the KPCLP header file supports three different data types u is for unsigned data types, I is for signed data types and f is for floating point data types.

The data size variable represents the size of the data that has been acquired from the Kinect camera in bits. Thus 8 is eight bit or one byte, 16 is sixteen bit or two bytes, 32 is thirty two bit of four bytes and 64 is sixty four bit or eight bytes.

The data dimensions variable represents the number of dimensions that each piece of data represents. For instance, a depth image has one data dimension whereas an RGB image has three.

The data resolution variable represents the resolution of the image received from the Kinect camera.

The data path variable represents the path to the binary encoded data that this KPCLP header file points to.

The epoch timestamp variable represents the amount of time that has passed since epoch at the point where the acquisition of the data that this KPCLP header file points to was acquired.

The Kinect timestamp variable represents the amount of time that has passed since the start of the acquisition of the data that this KPCLP header file points to was acquired.

The KPCLP header status variable currently means that the end of a KPCLP header file has been reached.

An example of a KPCLP file can be seen in the appendix bellow. (Appendix M:)

### Point Cloud Application: Overview

The point cloud application was mainly designed to fill the role of loading depth images and converting them into point clouds, saving and loading point clouds to file as PCD files, filtering by de noising and downsampling point clouds, registering a series of point clouds together, tracking a region of point cloud over time and outputting the results of the previous steps as an output file which could be simply parsed by another application.

To perform this role the point cloud application was linked to the PCL library this was in order to access methods which would allow for the saving and loading of point clouds to file as PCD files, filtering by de noising and downsampling point clouds and registering a series of point clouds together. The point cloud application was also linked

to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon these libraries.

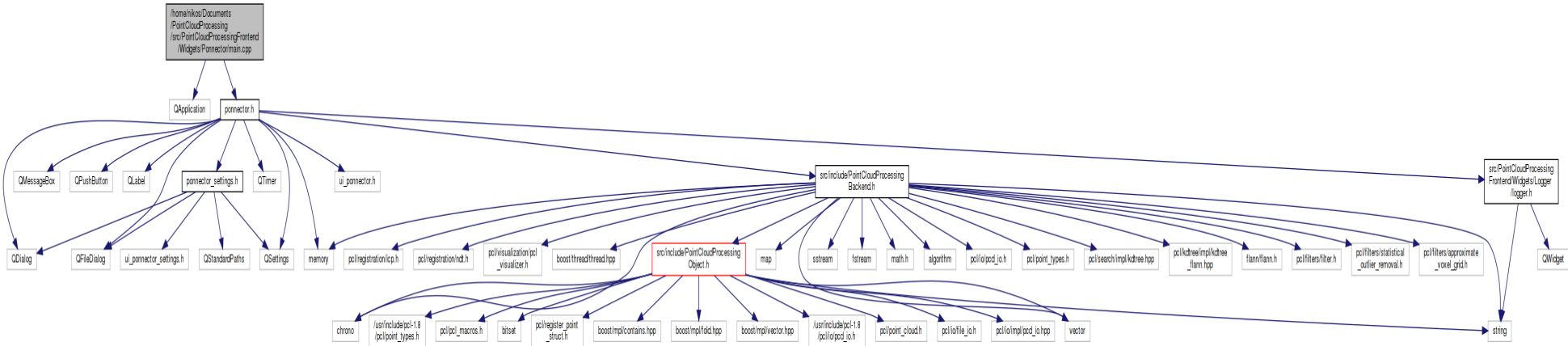


*Illustration 43: This illustration shows the unified modelling language (UML) class diagram for the entire point cloud application as a whole, this diagram shows the interaction between the classes of the point cloud application.*

The point cloud application contains five classes that are of interest. These classes are the Ponnector logger class, the point cloud back end class, the point cloud object class, the Ponnector class and the Ponnector settings class.

Most of the member variables contained within the classes of this application and some of their methods or functions are private. To be made private means to limit access to these variables and methods to anything from outside of the class that contains them.

The member variables and methods or functions of these classes have been made private in an attempt to follow the encapsulation method of programming, this means that the value or state of important structured data contained within the class is hidden from the user, or other applications, when they are outside of the class that encapsulates them. This is useful as it blocks unauthorised access to sensitive information held within these classes.

Encapsulation can also aid by removing complex helper functions from access outside of the class that uses it. This is useful because these helper functions have no use outside of the class in question so by being accessible they would only allow for the class in question to be broken or otherwise tampered with while only serving to clutter up the GUI. (Micallef, 1987)

**Point Cloud Application: Activity Diagram**

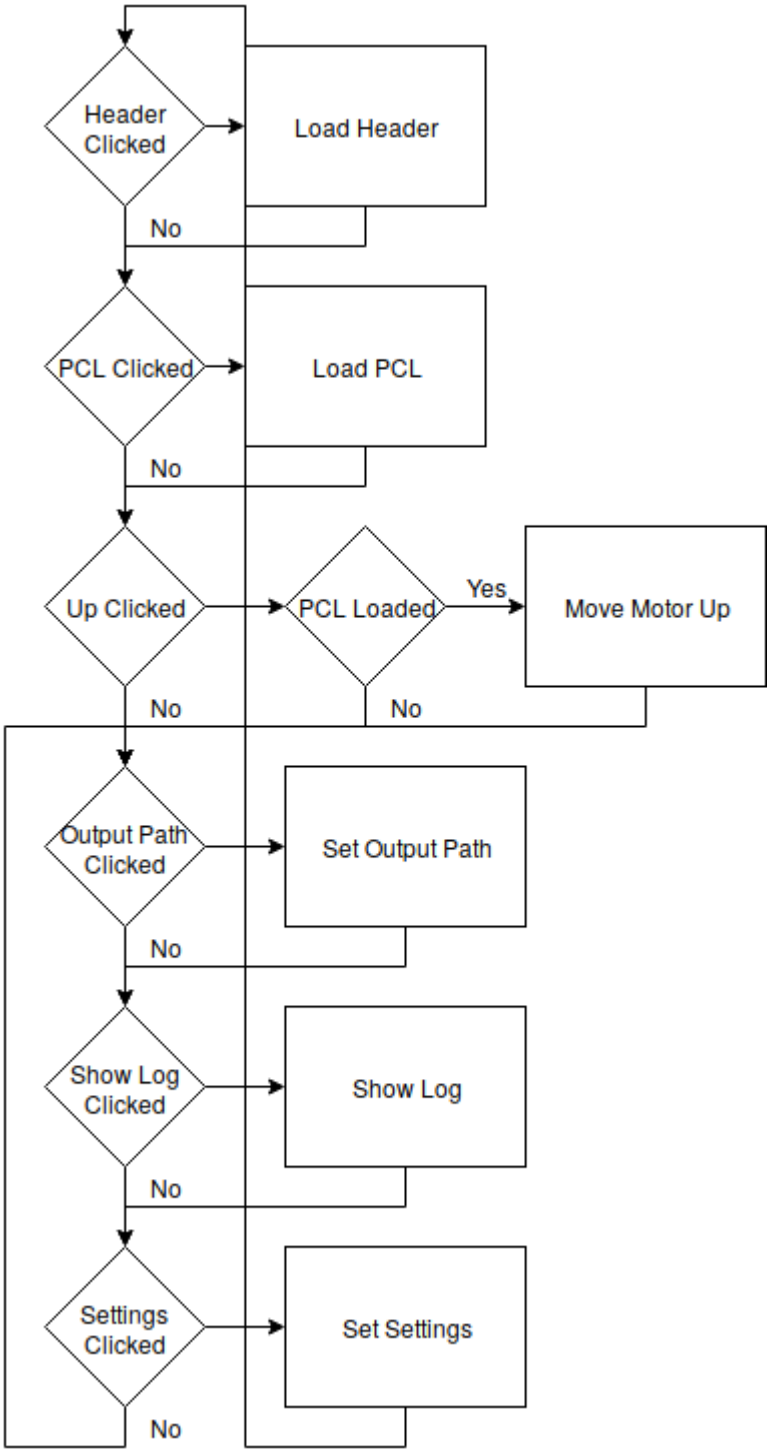


*Illustration 44: : This illustration shows the logical flow through the point cloud application as a UML activity diagram.*

**Point Cloud Application: Ponnector Logger Class**

The Ponnector logger class. This class is a Qt widget that is to be used in the Ponnector frontend class. This class can be used to output logs to the screen for the user to read.

The Ponnector logger class is associated with writing any output from the back end of the application to the screen.

To perform this role the Ponnector logger class was linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon this library.

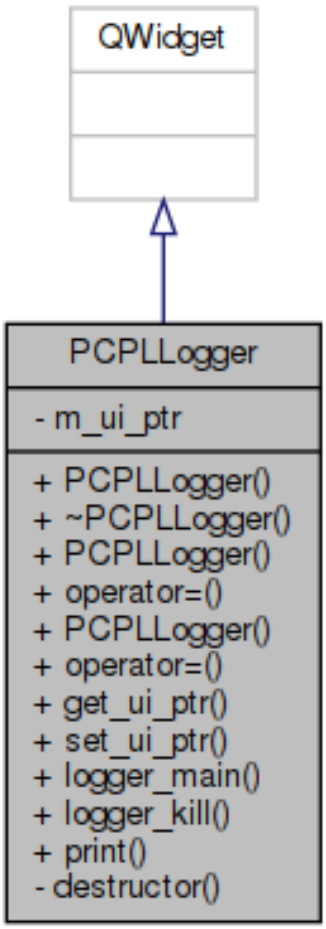


*Illustration 45: This illustration shows the UML class diagram for the Ponnector Logger class.*

The print method takes in a string of text and prints it to the screen.

The GUI pointer variable holds a pointer to the object that represents the Qt GUI widget of this class. This variable indicates that this is a Qt object.

### Point Cloud Application: Point Cloud Back End Class

The point cloud back end class. This class can, load headers, load data, output data as either a text and or binary file, calculate point clouds from depth images, output point clouds as PCD files either with text and or binary encoding, calculate the centre of gravity of a point cloud, calculate the difference between two centres of gravity, calculate signals from either two centres of gravity or from a tracked point cloud and can calculate the difference between two signals

The point cloud back end class is associated with loading depth images and converting them into point clouds, saving and loading point clouds to file as PCD files, filtering by de noising and downsampling point clouds, registering a series of point clouds together, tracking a region of point cloud over time and outputting the results of the previous steps as an output file which could be simply parsed by another application.

To perform this role the point cloud application was linked to the PCL library this was in order to access methods which would allow for the saving and loading point clouds to file as PCD files, filtering by de noising and downsampling point clouds and registering a series of point clouds together. Hence this application depends upon this library.

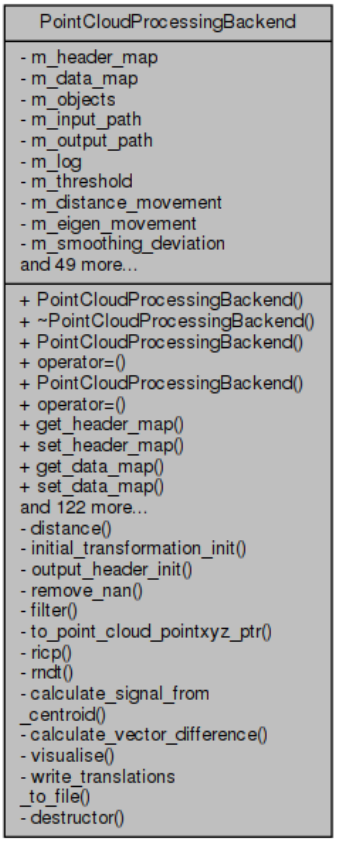


*Illustration 46: This illustration shows the UML class diagram for the point cloud back end class.*

The data enumeration is used as part of converting the KPCLP header files into point clouds. The enumeration contains the values u mapped to zero, I mapped to one and f mapped to two.

The header enumeration is used as part of converting the KPCLP header files into point clouds. The enumeration contains the values KPCLP header version mapped to zero, data type mapped to one, data size mapped to two, data dimensions mapped to three, data resolution mapped to four, data path mapped to five, epoch timestamp mapped to six, Kinect timestamp mapped to seven and KPCLP header status mapped to eight.

The calculate point cloud method calculates a point cloud from the data loaded from the KPCLP header file.

The calculate signal from centroid method is a private method which takes in a pointer to a point cloud object and calculates a region from this point cloud based on the signal settings and outputs this as a four dimensional (4D) vector.

The calculate vector difference method is a private method which takes in two different 4D vectors and outputs the Euclidean linear distance between them.

The distance method is a private method which takes in six floating point numbers, which represent two 3D vectors, and outputs the Euclidean linear distance between the two 3D vectors.

The filter method is a private method which takes in a pointer to a point cloud object and applies SOR and VGF to this point cloud based on the filtering settings.

The initial transformation initialisation method is a private method which returns the initial guess matrix.

The load data method loads the data located at the path in the KPCLP header file.

The load headers method takes in a array or linked list of paths to KPCLP header files and loads and sorts the KPCLP header files at these paths.

The load PCD method takes in a array or linked list of paths to PCD files and loads the PCD files at these paths.

The output header initialisation method is a private method which initialises the output file based on the settings of the current acquisition.

The register method iterates through all of the point clouds stored in memory and registers them to each other based on the registration settings.

The remove NAN method is a private method which removes all NAN values from a point cloud.

The ICP method is a private method which takes in a source point cloud, a target point cloud and a guess matrix, applies ICP registration to register the source point cloud to the target point cloud and then returns the transformation calculated and the error between the two point clouds.

The NDT method is a private method which takes in a source point cloud, a target point cloud and a guess matrix, applies NDT registration to register the source point cloud to the target point cloud and then returns the transformation calculated and the error between the two point clouds.

The vector to point cloud method is a private method which takes in a 4D vector and returns the same 4D vector placed into the first element of a point cloud.

The visualise method is a private method which takes in three point clouds and three centres of gravity and visualises them to the user.

The write point cloud to file method which writes the point clouds in memory to file using text and or binary encoding.

The write output to file method is a private method which takes in a string of text and writes it to file using text encoding.

The point cloud back end class contains variables which are set in the settings section of the application and serve only to update the application based on the users input. These include variables which represent the input and output path of the application and the things to be output from the application. This class also provides variables which represent if the application should be run in test mode, if the application should output visualisations of its output, the size and colour of the objects in this visualisation, the type of registration to use, the object to be registered to, the algorithm to use to detect the distance between two point clouds, the distance to the global threshold, the variables used during the depth image acquisition, the threshold at which movement is deemed to be considered great enough to warrant registration, the variables used in the filtering algorithms, the tracking algorithm to be used, the size and position of the object to be tracked, the type of guess translation to be used in the registration algorithm and the initial guess translation.

The data map variable is a map which represents the data types in the KPCLP header file.

The header map variable is a map which represents the string of text in the KPCLP header file.

The objects variable is an array or linked list of pointers to point cloud object objects which are used to represent the point clouds which are to be registered.

### Point Cloud Application: Point Cloud Pseudo Code

*x = (i - width / 2) * (depth + offset) * focalLength*

*y = (j - height / 2) * (depth + offset) * focalLength*

*z = depth*

*Where*

*offset = -10*

*focalLength = .0021*

The above calculation shows how to calculate one point in a point cloud from a depth in a depth image. This calculation was sourced from the OpenKinect Libfreenect documentation and the offset and focal length used here are generic and represent the values of an average camera. The values above have an accuracy of plus or minus three millimetres at two metres of depth in the horizontal and vertical dimensions and an accuracy of plus or minus two millimetres at two metres of depth in the depth dimension. (*Imaging Information - OpenKinect*, no date)

```
Offset equals offset from settings

Focal length equals focal length from settings

Load depth images from KPCLP header file

For each depth images

    For width

        For height

            Point cloud at index vertical component equals width index minus half width multiplied by depth image at index plus offset multiplied by focal length

            Point cloud at index horizontal component equals height index minus half height multiplied by depth image at index plus offset multiplied by focal length

            Point cloud at index depth component equals depth image at index
```

The above pseudo code shows the process by which a point cloud is calculated from a depth image. This only shows one thread of logic and it assumes that loading a depth image from a KPCLP header file and using a threshold and inserting NAN variables is trivial.

First, the relevant variables to control the flow of logic through the code are set from the variables in the application settings.

Then, for each depth image loaded from a KPCLP header file a point cloud is calculated point by point using the equation from the OpenKinect libfreenect documentation.

### Point Cloud Application: Registration Pseudo Code

```
Distance equals distance from settings
```

```
Registration equals registration from settings
Output equals output from settings
For each point cloud
    Remove NAN values from point cloud
    Apply de noising algorithm to point cloud
    Downsample point cloud
    Find centre of gravity of point cloud
    If distance from centre of gravity to previous centre of gravity is greater than distance
        Tracking
        If Registration equals ICP
            Register point cloud to previous point cloud using ICP
        else
            Register point cloud to previous point cloud using NDT
        Store data calculated from tracking and registration
If output equals true
    Output data calculated from all tracking and registrations
```

The above pseudo code shows the process by which point clouds are tracked and registered to each other. This only shows one thread of logic and it assumes that the tracking algorithms and the registration algorithms from the PCL are trivial.

First, the relevant variables to control the flow of logic through the code are set from the variables in the application settings.

Then, for each point cloud in memory the application removes all NAN variables which have been selected based on the perspective of the camera or any value that falls behind the threshold.

Next, the filtering algorithms are applied to each point cloud, this includes the SOR and VGF algorithms. The variables of which are set from the settings section of the application.

After filtering the centre of gravity of the point cloud is calculated. This centre of gravity is then used to estimate the average movement between this point cloud and the

previous point cloud, if the estimated distance is less than the distance value from the settings of the application then the registration step is skipped for this point cloud.

Finally, the tracking and registration algorithms are applied to this point cloud and the previous point cloud, the type of tracking and registration is selected using the application settings. The output from these algorithms like the homogeneous transformations between the point clouds are stored in memory to be output to file later.

### Point Cloud Application: Tracking Pseudo Code

```
Signal centre equals signal centre from settings

Signal radius equals signal radius from settings

Find centre of gravity of first point cloud

Find point at signal centre away from centre of gravity of first point cloud

Tracking point cloud equals all points at signal radius away from point at signal centre away from centre of gravity of first point cloud

For each point cloud

    If Registration equals ICP

        Register tracking point cloud to point cloud using ICP

    else

        Register tracking point cloud to point cloud using NDT

    Store data calculated from tracking and registration
```

The above pseudo code shows the process by which a tracking point clouds is tracked or registered to every subsequent point cloud. This only shows one thread of logic and it assumes that the registration algorithms from the PCL are trivial.

First, the relevant variables to control the flow of logic through the code are set from the variables in the application settings.

Then, the tracking point cloud is calculated from the first point cloud. This is done be first finding the centre of gravity of the first point cloud and from this position moving the distance away specified by the signal centre variable from the settings of the application. Once at this new position every point which is the signal radius away is stored in the tracking point cloud, thus the tracking point cloud becomes a spherical ROI.

Finally, the tracking and or registration algorithms are applied to this tracking point cloud and every subsequent point cloud, the type of registration is selected using the application settings. The output from this algorithm, for instance the homogeneous transformations between the tracking point cloud and the given point cloud are stored in memory to be output to file later.

### Point Cloud Application: Point Cloud Object Class

The point cloud object class. This class represents the objects loaded from the KPCLP header files and the PCD files.

The point cloud object class is associated with holding the data of a given KPCLP header file and the point cloud which is calculated from the data of a given KPCLP header file.

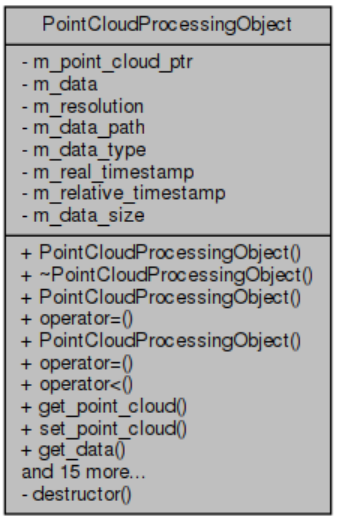


*Illustration 47: This illustration shows the UML class diagram for the point cloud object class.*

The less than operator is overloaded with a method that returns the comparison of the real timestamp of two point cloud object objects.

The data variable is an array or linked list that holds the depth images loaded from the KPCLP header file.

The data path variable holds the path to the data that the KPCLP header file points to.

The data size variable holds the size of each piece of data in the file that the KPCLP header file points to.

The data type variable holds the data type of the data that the KPCLP header file points to.

The point cloud pointer variable holds a pointer to a point cloud object that has been calculated from the data that he KPCLP header file points to.

The real timestamp variable holds the number of milliseconds that had passed since epoch when the data that the KPCLP header file points to was acquired.

The relative timestamp variable holds the number of milliseconds that had passed since the Kinect camera started the acquisition when the data that the KPCLP header file points to was acquired.

The resolution variable is an array or linked list that holds the resolution of the data that the KPCLP header file points to.

### Point Cloud Application: Ponnector Class

The Ponnector class. This class is a Qt frontend for the point cloud back end classes. This class calls the functions of the point cloud back end classes, collects the output from these classes and then displays this output to the user.

The Ponnector class is associated with providing a GUI solution to the user in order to allow them to interact with the application.

To perform this role the Ponnector class was linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon this library.

This class provides event handlers that call functions to load a given set of KPCLP header files into the applications, to either convert the data of a given set of KPCLP header files into point clouds or to load a given set of PCD files into the application and to register the point clouds which are loaded into a buffer at that given time.

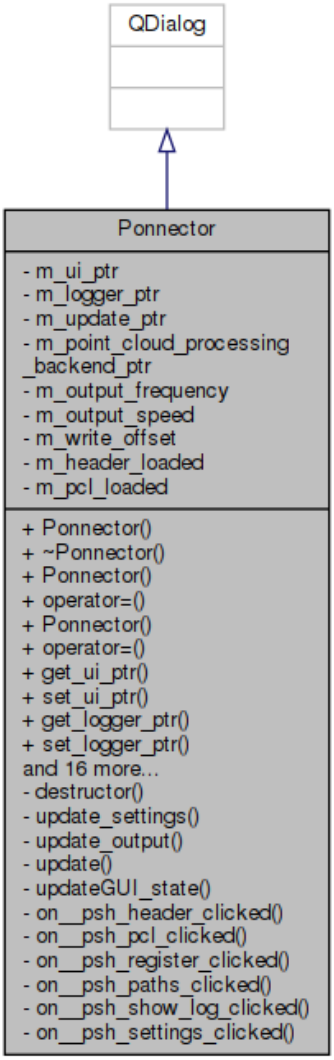


*Illustration 48: This illustration shows the UML class diagram for the Ponnector class.*

The header method is a private slot method which reacts to the event of the header button being pressed. This method loads the KPCLP header files.

The paths method is a private slot method which reacts to the event of the paths button being pressed. This method changes the input and output paths.

The PCL method is a private slot method which reacts to the event of the PCL button being pressed. This method either converts the data loaded from the KPCLP header file into point clouds or loads point clouds from PCD files.

The register method is a private slot method which reacts to the event of the register button being pressed. This method registers the point clouds which are currently stored in memory.

The settings method is a private slot method which reacts to the event of the settings being pressed. This method changes the settings of the application.

The show log method is a private slot method which reacts to the event of the show log button being pressed. This method shows the log.

The Ponnector main method is used to initialise the Qt GUI.

The update method is a private slot method which reacts to a timer elapsing at a periodic rate. This method calls the update method of the Ponnector logger class.

The update output method is a private method which updates the GUI.

The update settings method is a private method which updates the settings of the application.

The update GUI state is a private slot method which tracks the current state of the GUI.

The header loaded variable is true if a KPCLP header file has been loaded.

The Ponnector logger pointer variable holds a pointer to the GUI object that writes text to the screen.

The output frequency variable holds the frequency at which files are to be output.

The output speed variable holds the number of milliseconds that need to elapse before another output can take place.

The pcl loaded variable is true either if a PCD file has been loaded or if point clouds have been calculated from the data that the KPCLP header file points to.

The point cloud back end pointer variable holds a shared pointer to the object that deals with the back end of the application.

The GUI pointer variable holds a pointer to the object that represents the Qt GUI widget of this class. This variable indicates that this is a Qt object.

The update pointer variable holds a pointer to the object that times the duration between updates.

The write offset variable holds the amount of time that must elapse before the GUI can be updated.

**Point Cloud Application: Use Case Diagram**

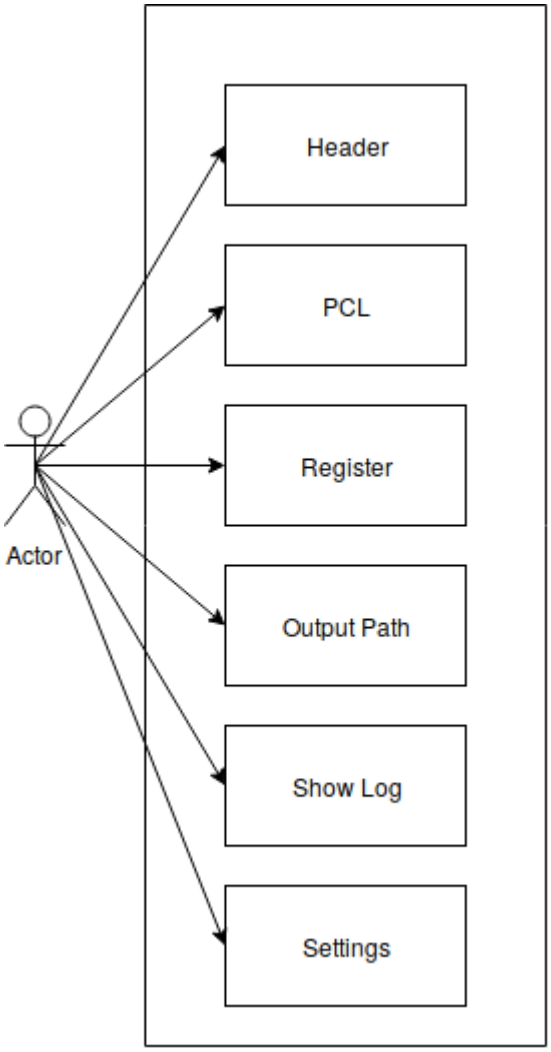


*Illustration 49: : This illustration shows the options that the user has in the point cloud application as a UML use case diagram.*

**Point Cloud Application: Graphical User Interface**

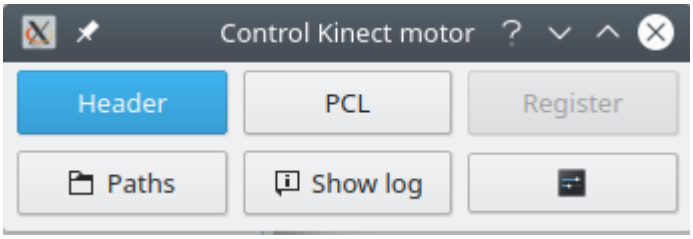


*Illustration 50: This illustration shows the GUI of the point cloud application as it appears when the application is initially started. All options are blocked except for loading a KPCLP header file, loading a PCD file and adjusting the settings and opening other Qt widgets.*

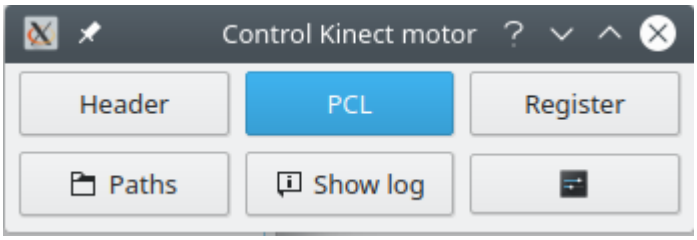


*Illustration 51: This illustration shows the GUI of the point cloud application as it appears after a KPCLP header file has been loaded. The register option is now available to be interacted with.*

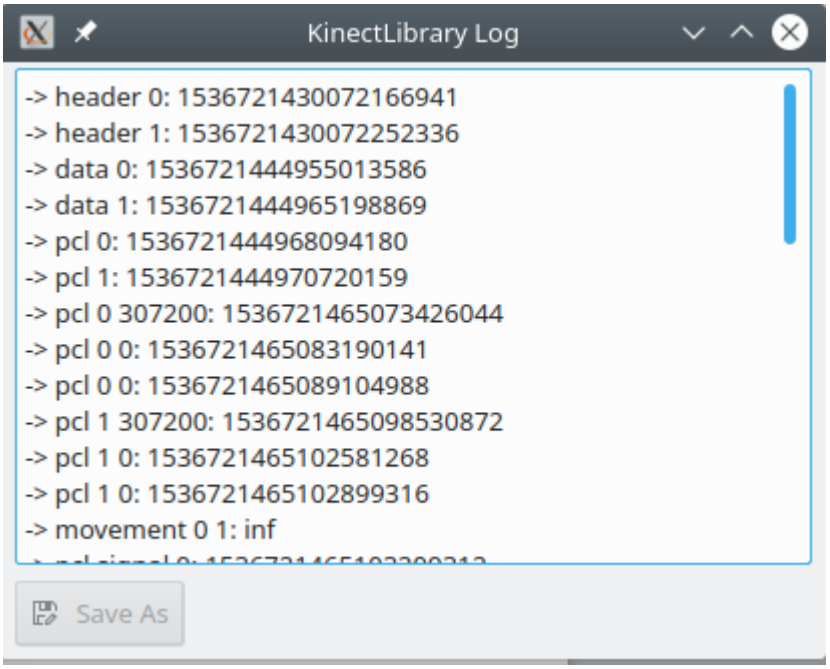


*Illustration 52: This illustration shows the GUI of the point cloud logger application as it appears after a KPCLP header file has been loaded, converted into a point cloud and then registered. Arrows are used to show input and output from the application.*

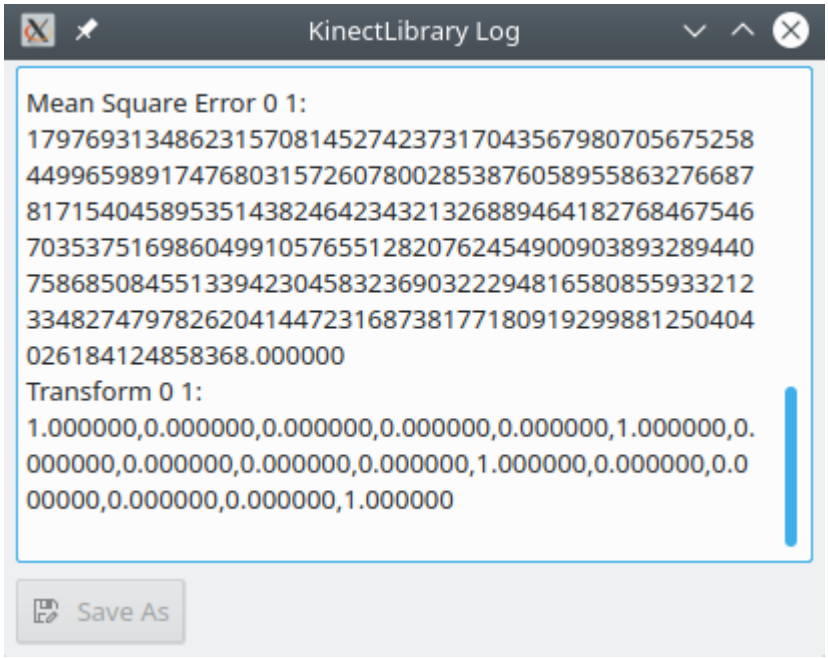


*Illustration 53: This illustration shows the GUI of the Kinect logger application as it appears after connecting to a Kinect camera. Arrows are used to show input and output from the application.*

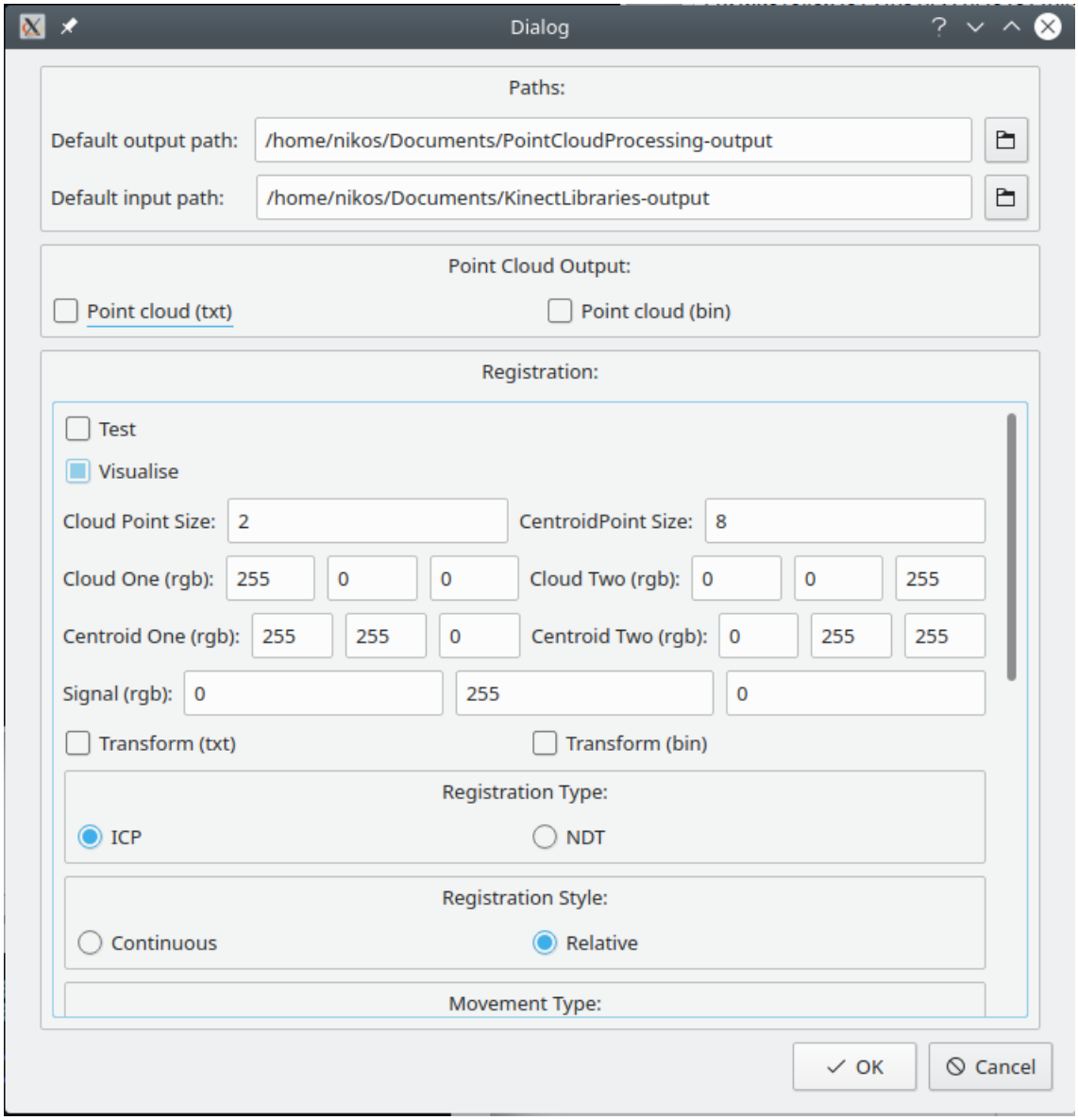


*Illustration 54: This illustration shows the GUI of the Kinect settings application.*

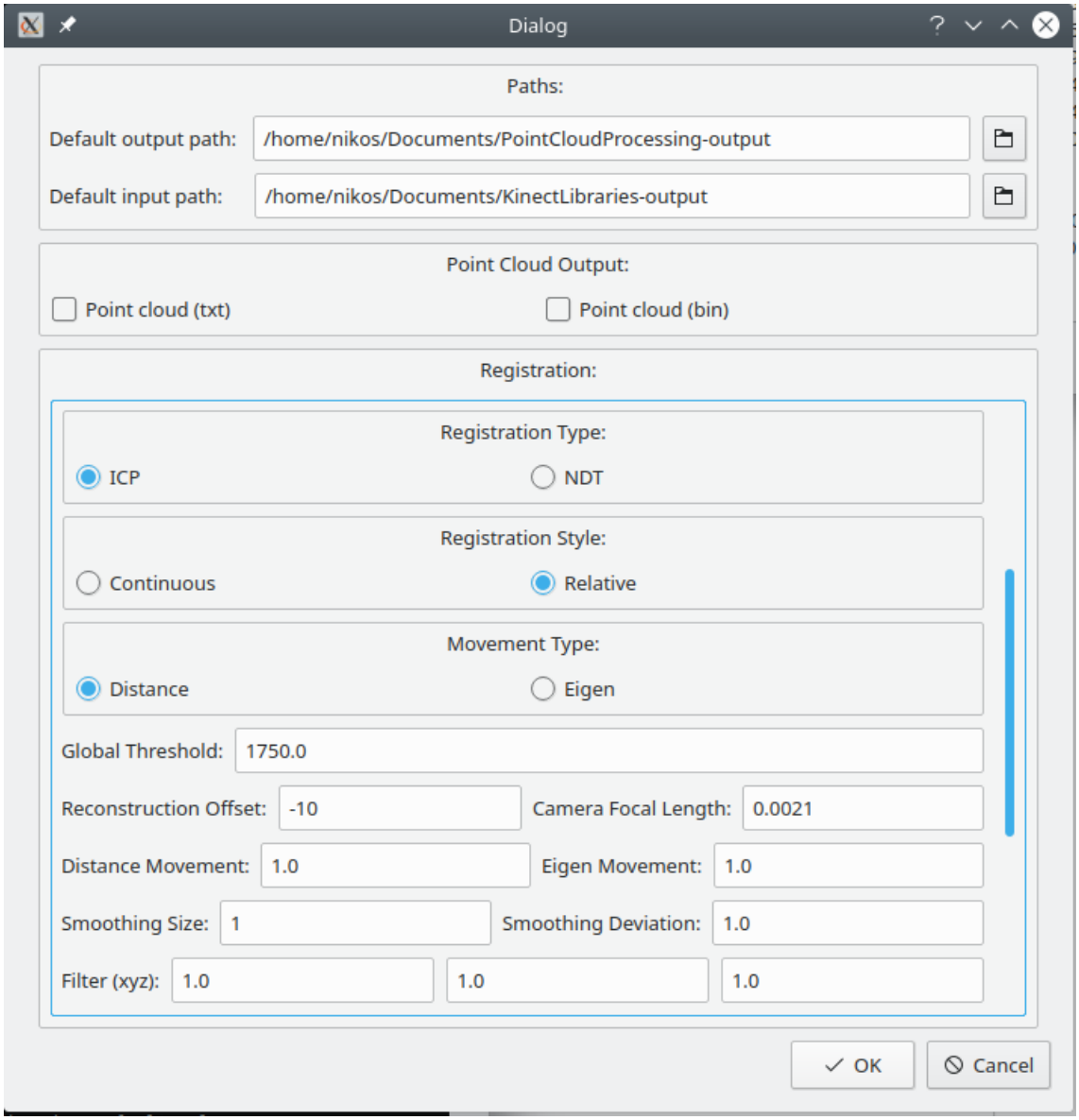


*Illustration 55: This illustration shows the GUI of the Kinect settings application.*

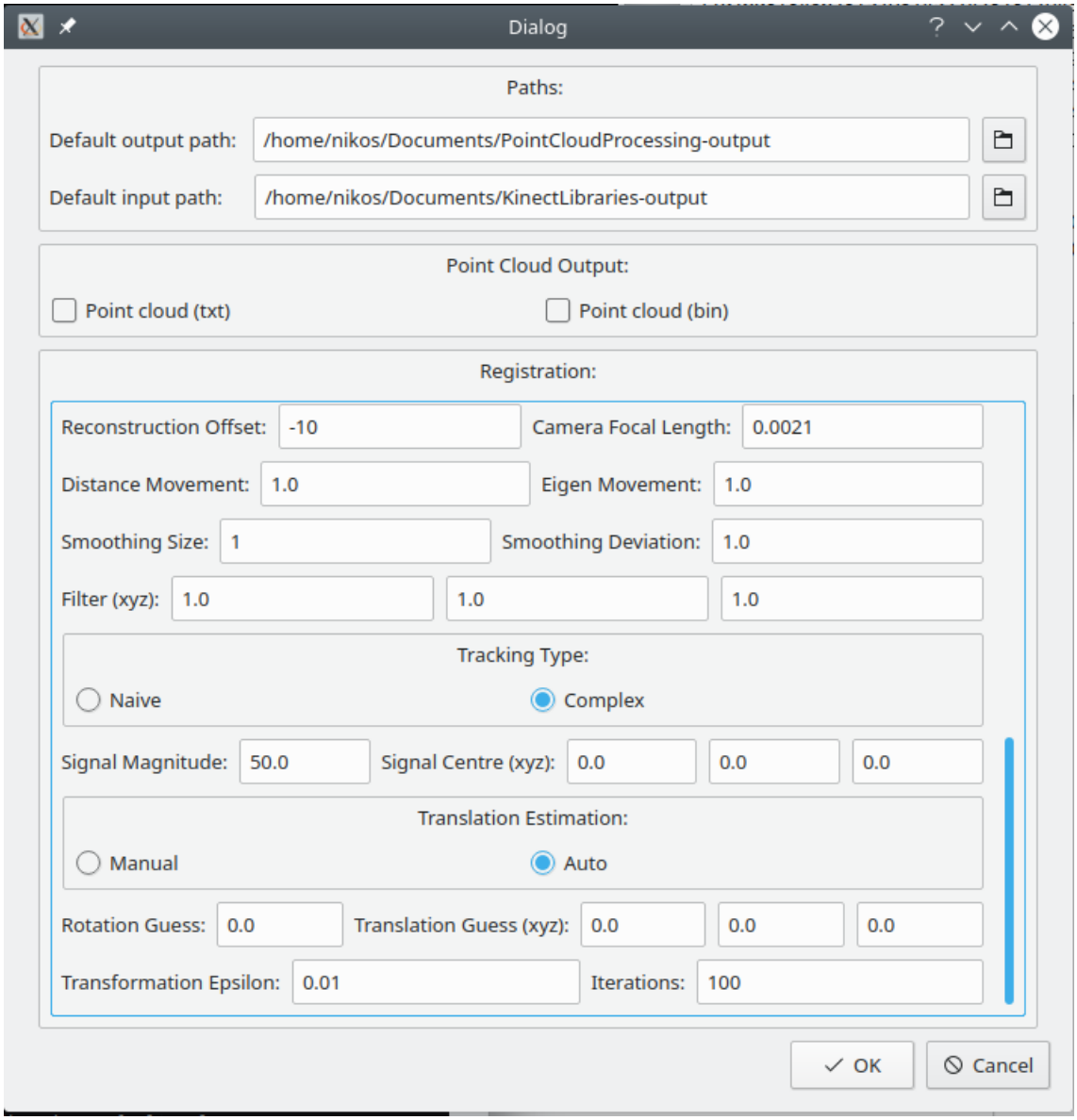


*Illustration 56: This illustration shows the GUI of the Kinect settings application.*

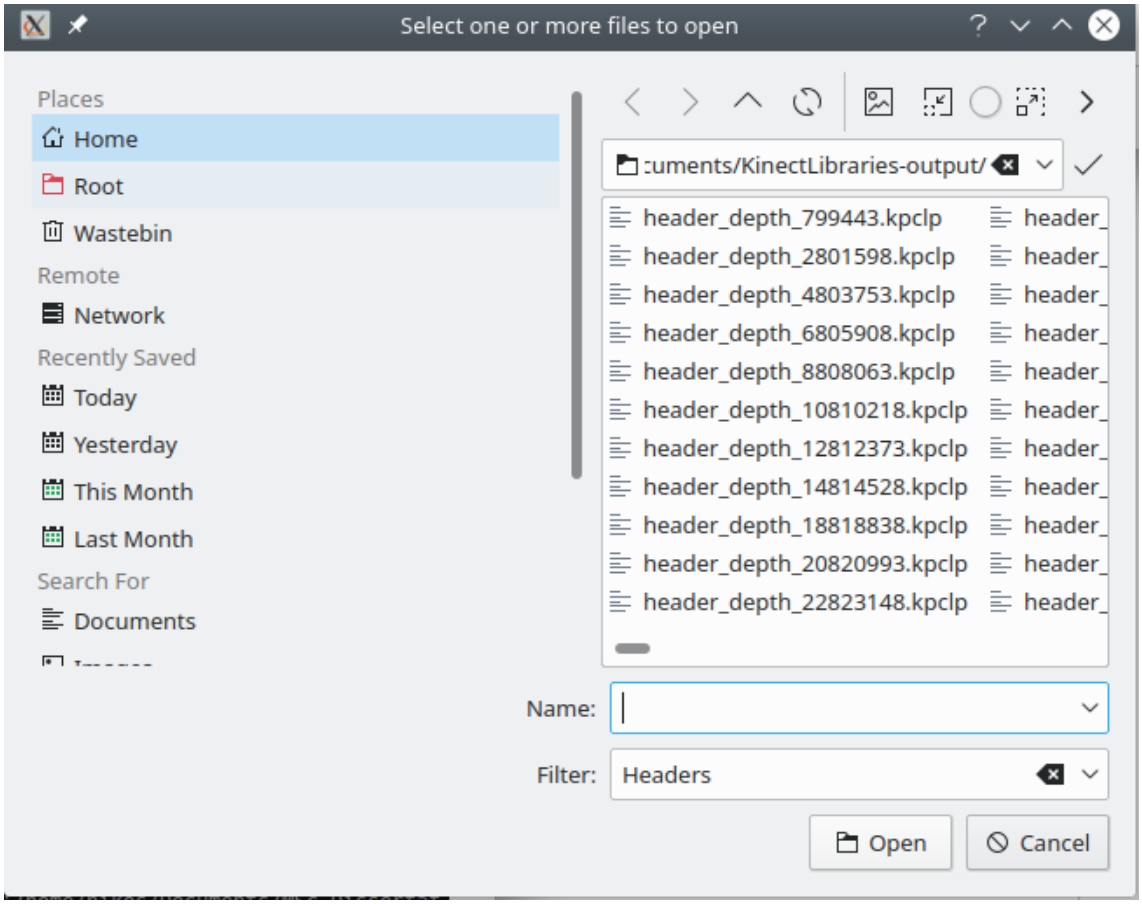


*Illustration 57: This illustration shows the Qt widget used to select paths.*

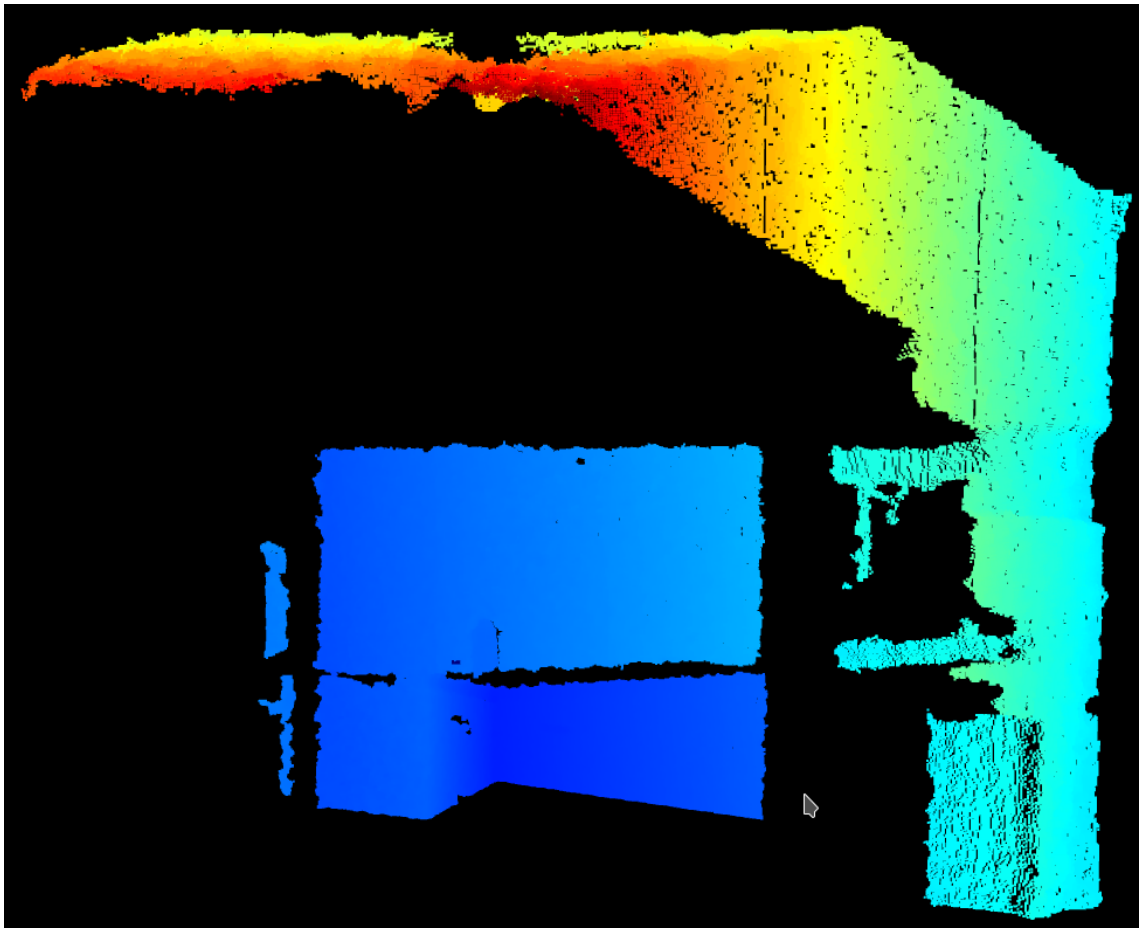

*Illustration 58: This illustration shows an RGB point cloud visualisation.*

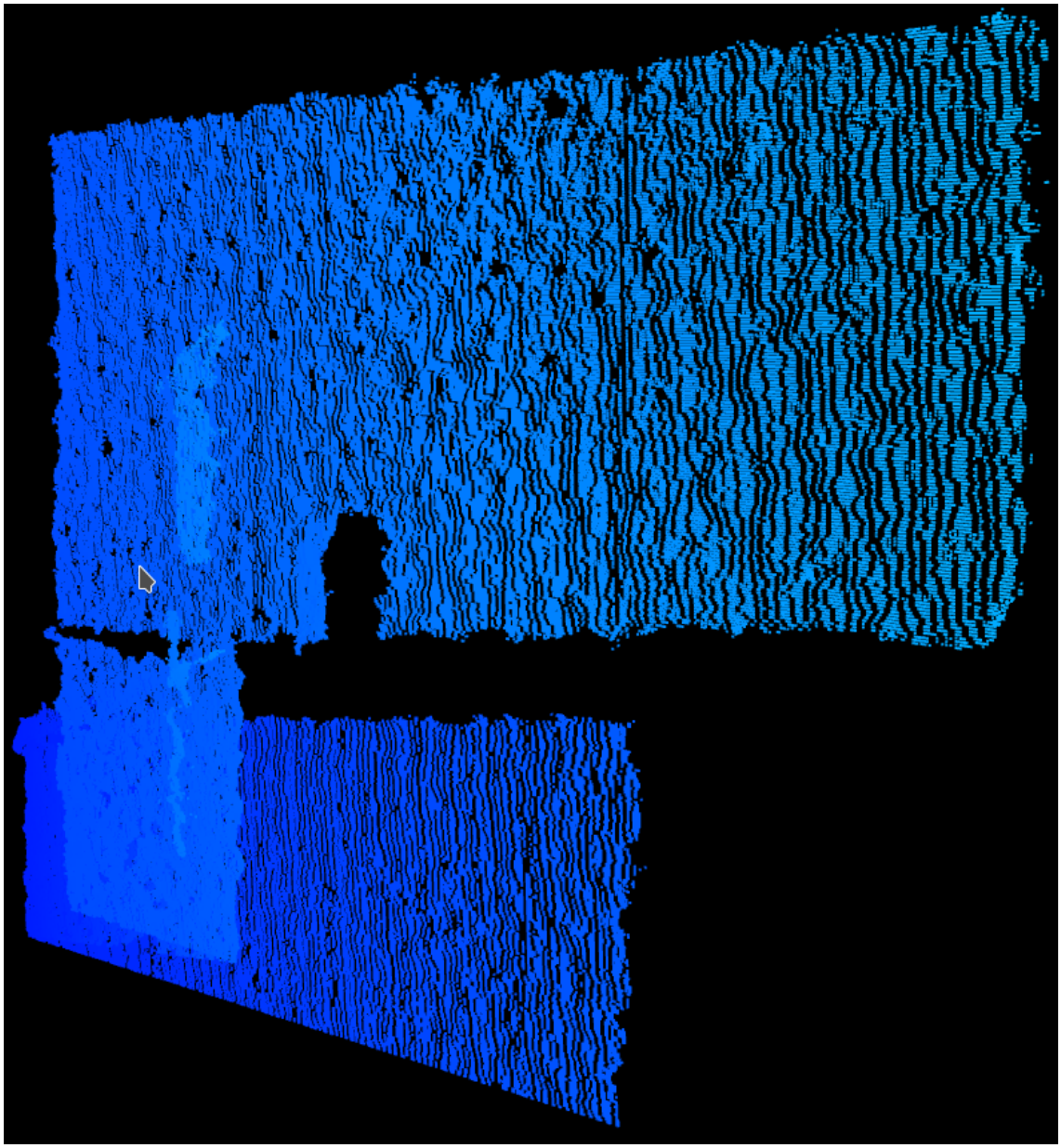

*Illustration 59: This illustration shows a close up of the above visualisation (Illustration 58).*

*Illustration 60: This illustration shows an RGB point cloud visualisation.*

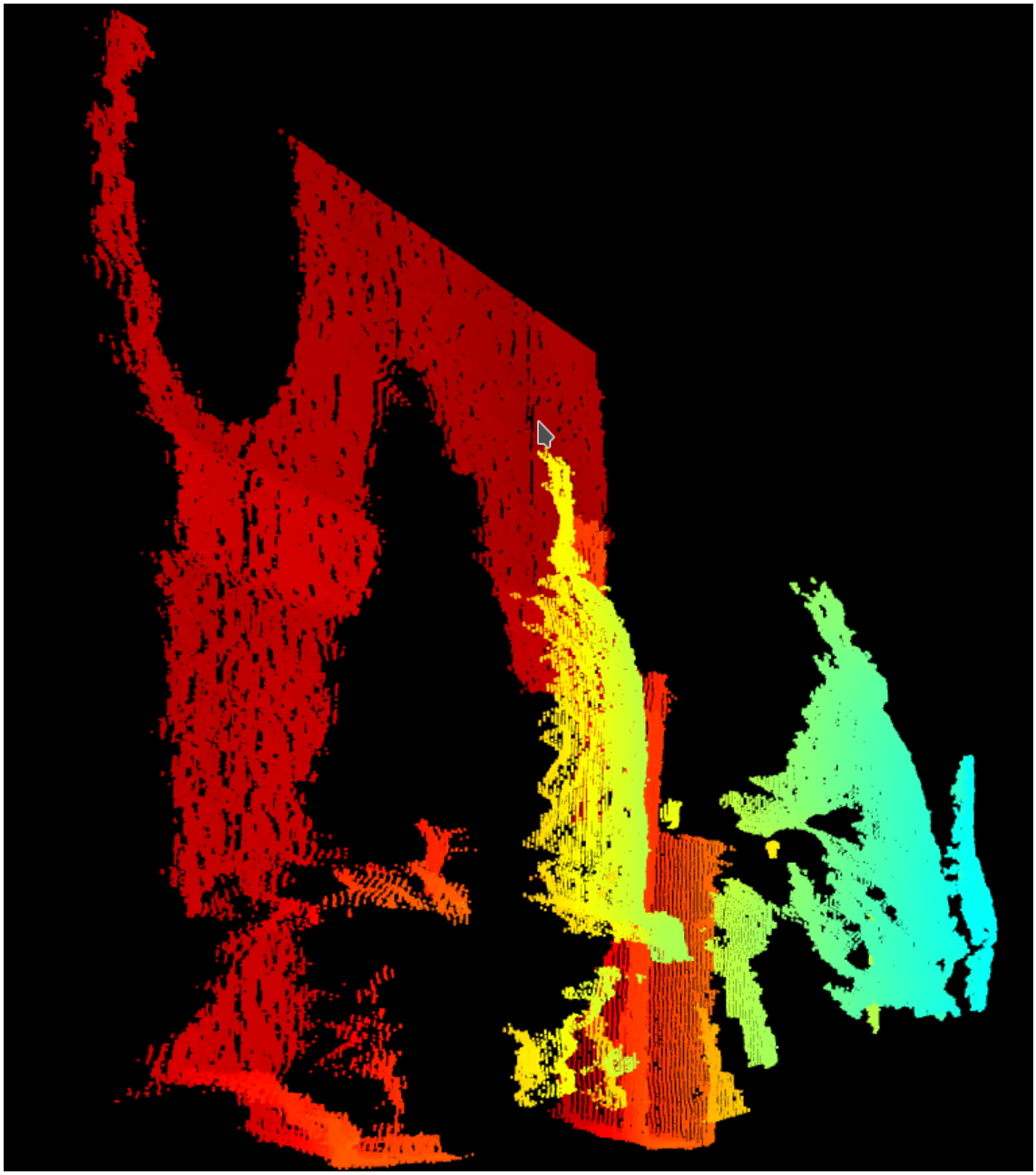

*Illustration 61: This illustration shows a close up of the above visualisation (Illustration 60).*

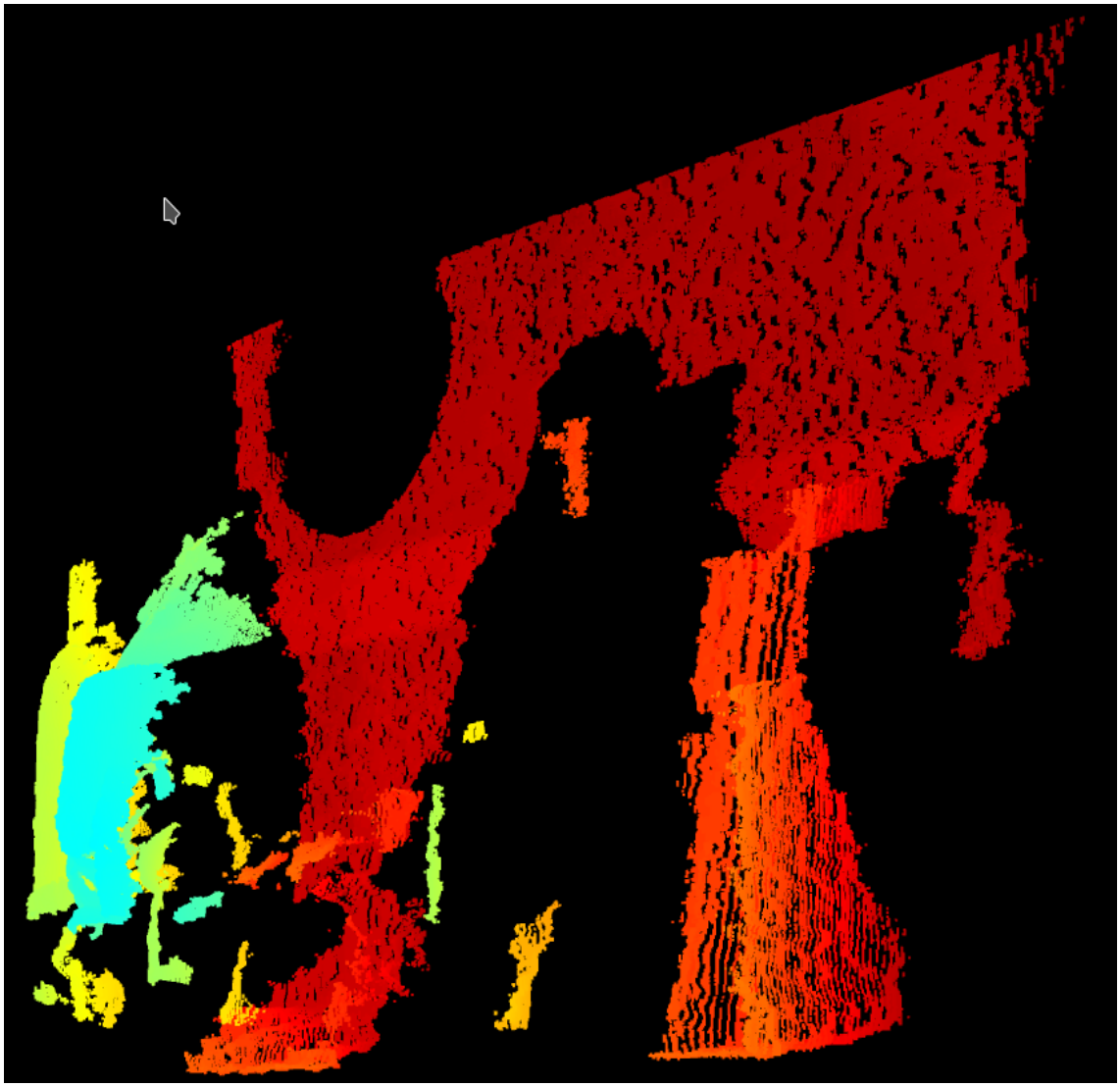

*Illustration 62: This illustration shows a close up of the above visualisation (Illustration 60).*

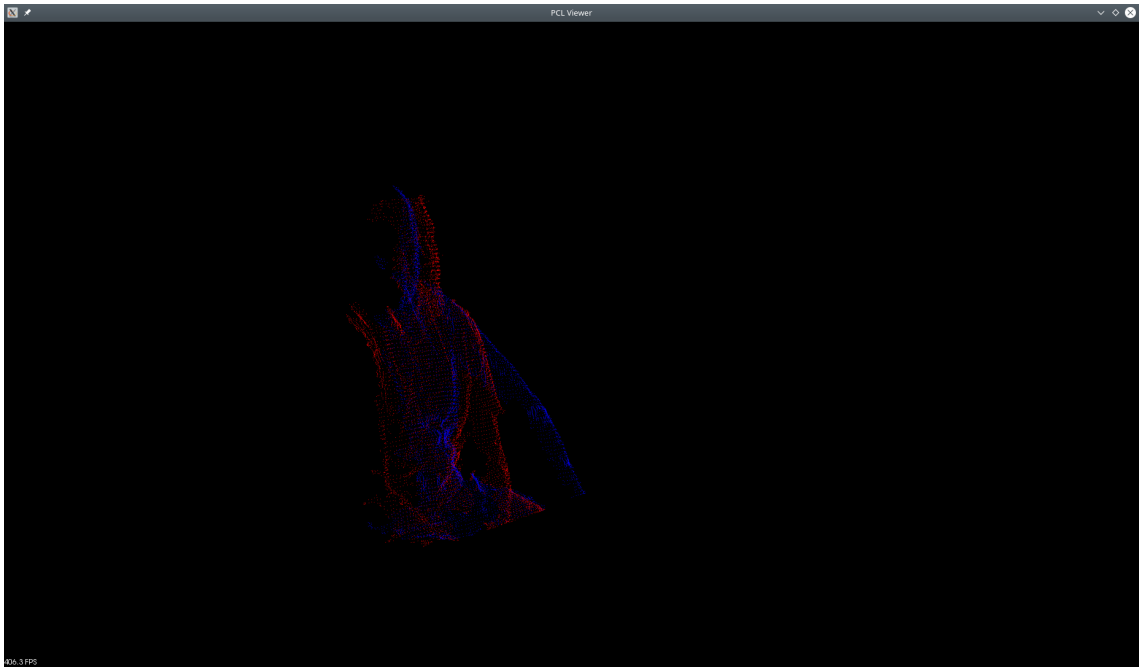


*Illustration 63: This illustration shows two monochrome point clouds which have been registered to each other.*

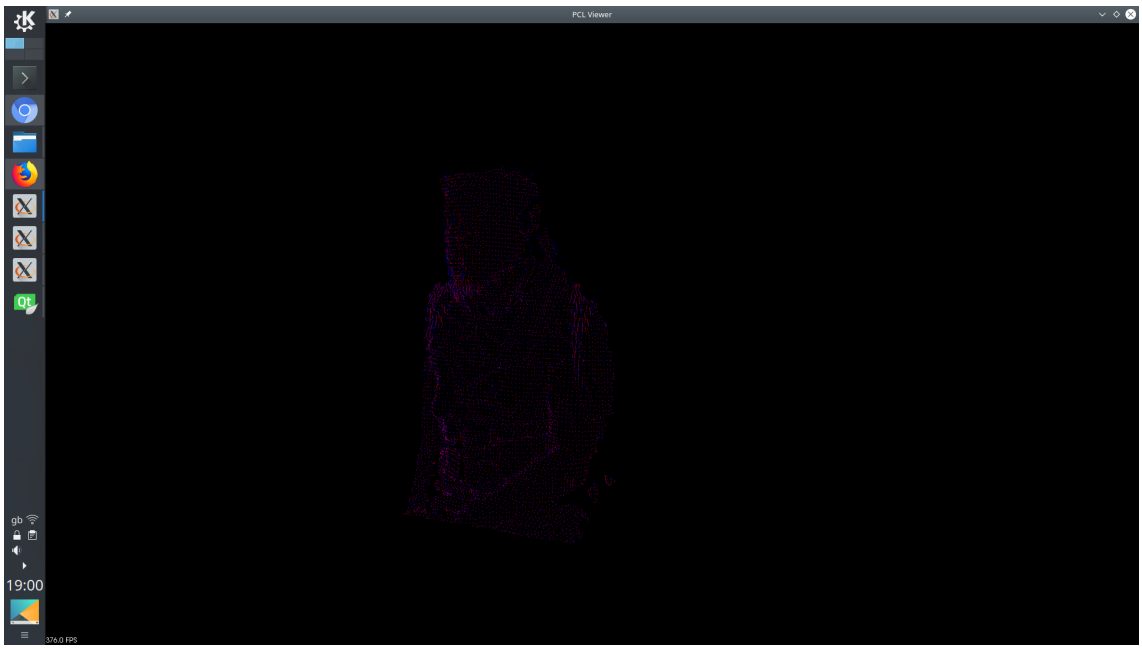


*Illustration 64: This illustration shows two monochrome point clouds which have been registered to each other.*

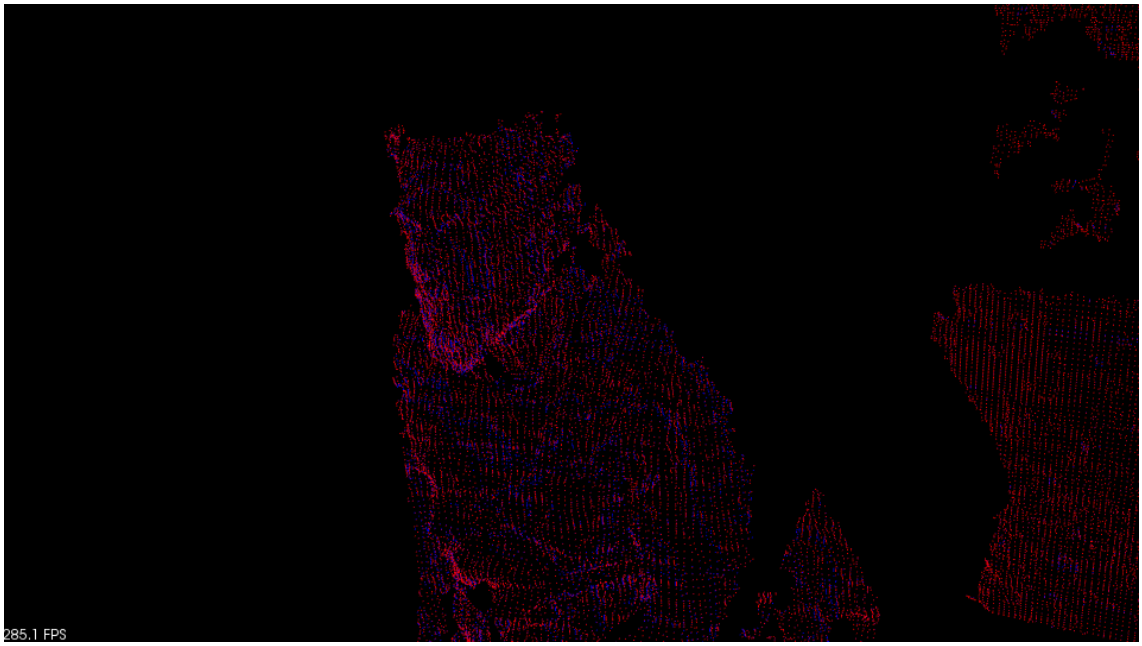


*Illustration 65: This illustration shows two monochrome point clouds which have been registered to each other.*

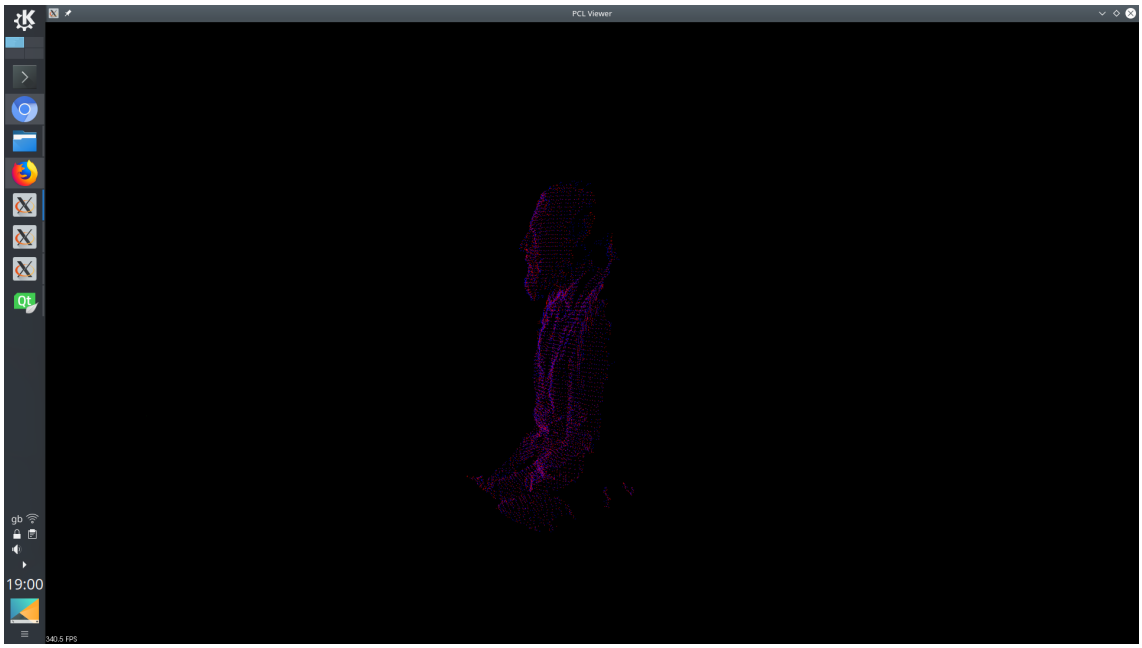


*Illustration 66: This illustration shows two monochrome point clouds which have been registered to each other.*

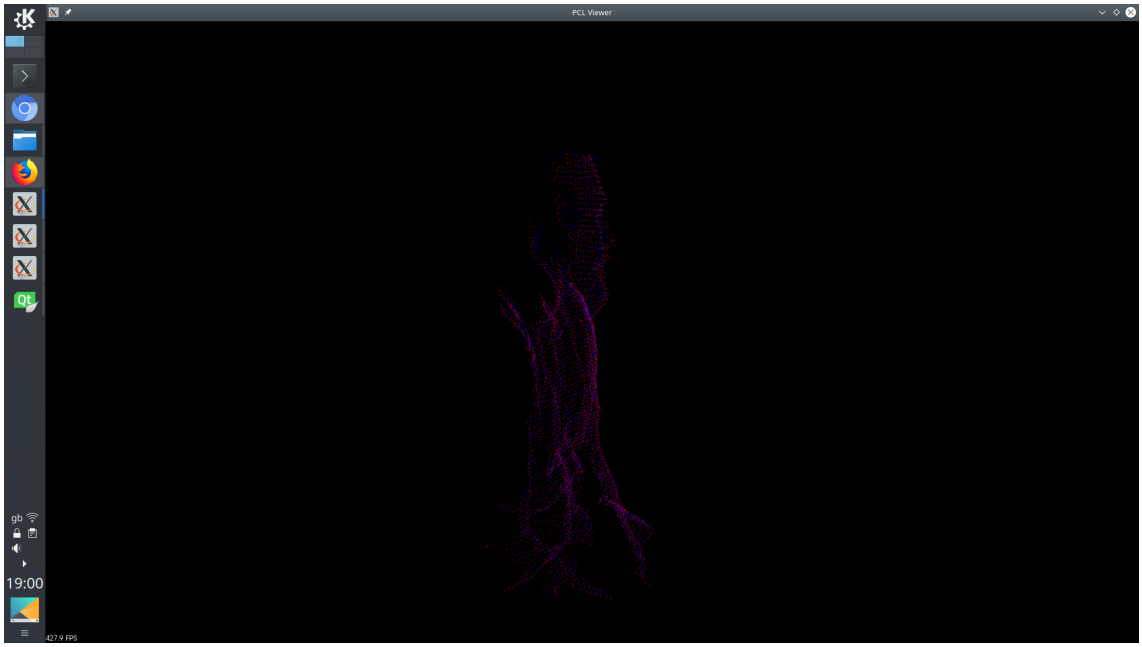


*Illustration 67: This illustration shows two monochrome point clouds which have been registered to each other.*

### Point Cloud Application: Ponnector Settings Class

The Ponnector Settings class. This class holds the settings that are to be used in the Ponnector frontend class.

The Ponnector settings class is associated with providing a GUI solution to the user in order to allow them the adjust the settings of the application.

To perform this role the Ponnector settings class was linked to the Qt GUI toolkit in order to allow for the development and deployment of a cross platform GUI solution. Hence this application depends upon this library.

This class provides variables which can store the input and output path of the application and the things to be output from the application. This class also provides variables which represent if the application should be run in test mode, if the application should output visualisations of its output, the size and colour of the objects in this visualisation, the type of registration to use, the object to be registered to, the algorithm to use to detect the distance between two point clouds, the distance to the global threshold, the variables used during the depth image acquisition, the threshold at which movement is deemed to be considered great enough to warrant registration, the variables used in the filtering algorithms, the tracking algorithm to be used, the size

and position of the object to be tracked, the type of guess translation to be used in the registration algorithm and the initial guess translation.

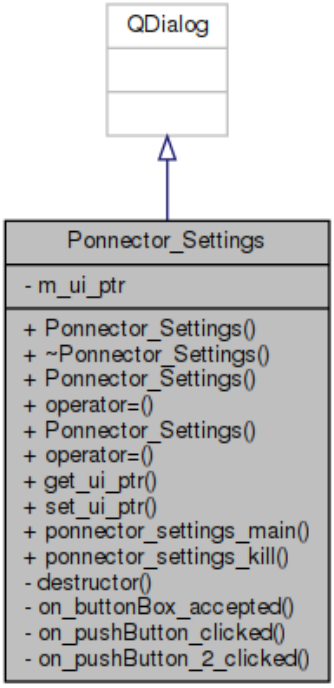


*Illustration 68: This illustration shows the UML class diagram for the Ponnector settings class.*

The button method is a private slot method which reacts to the event of the button being pressed. This method updates the saved settings.

### Point Cloud Application: Output

The output file holds all of the data which has been calculated while the point cloud application has been running. Each output file starts with a header which details all of the settings that were used when the registration algorithm was run to produce that particular output file.

The registration type variable represents the type of registration algorithm which was used. Currently the point cloud application supports both ICP and NDT.

The registration style variable represents how the point clouds were selected for registration. Currently the point cloud application supports continuous and relative point cloud selection. A continuous selection is where the sequence of selection would proceed zero to one, one to two, two to three etc. A relative selection is where the sequence of selection would proceed zero to one, zero to two, zero to three etc.

The movement type variable represents the way that the amount of movement between two point clouds was calculated. Currently the point cloud application supports Euclidean linear distance and Eigenvectors.

The movement distance variable represents the amount of movement that must be calculated as occurring between two point clouds before they will be registered to each other.

The translation estimation variable represents if the output of the previous registration step was used as the input of the next registration step.

The tracking type variable represents the tracking algorithm which was used to estimate the surrogate or breathing signal. Currently the point cloud application supports Euclidean linear distance and tracking the registration of a feature of the point cloud

The threshold variable represents the cut off for where depth information is ignored when calculating a point clouds.

The offset variable represents the offset which was used when calculating the point clouds.

The focal length represents the focal length which was used when calculating the point clouds.

The smoothing size represents the number of point included in the SOR algorithm when calculating the standard deviation.

The smoothing deviation represents the cut off for what is considered noise in the SOR algorithm

The filter (xyz) variables represents the size of the voxel filters used in the VGF algorithm.

The transformation epsilon variable represents the cut off at which point the registration algorithm considered two point clouds as aligned.

The iterations variable represents the maximum number of iterations that the registration algorithm will be allowed to attempt.

The rotation guess variable represents the initial rotation of the guess matrix.

The translation guess (xyz) variables represent the initial translations of the guess matrix.

The signal magnitude represents the radius of the region that is used in the tracking algorithm.

The signal (xyz) variables represent the position of the centre of the region that is used in the tracking algorithm.

The file paths variables represent the position of the point clouds used for that particular registration step and their file paths.

The registration variable represents the position of the point clouds used for that particular registration step and whether that registration step was skipped or not.

The source centroid variable represents the centre of gravity of the first point cloud.

The target centroid variable represents the centre of gravity of the second point cloud.

The centroid differences variable represents the Euclidean linear distance between the centres of gravity of the two point clouds.

The signal variable  represents the position of the point clouds used for that particular tracking step.

The signal mean square error variable represents the difference between the positions of the two point clouds post tracking.

The signal transform variable represents the homogeneous transformation between the two point clouds post tracking.

The mean square error variable represents the difference between the positions of the two point clouds post registration.

The transform variable represents the homogeneous transformation between the two point clouds post registration.

An example of an output file can be seen in the appendix bellow. (Appendix N:)

### Test Application: Overview

The test application was mainly designed to fill the role of producing test data to use to verify the integrity of the point cloud application. This test data can be generated of any size, either resolution of depth and can be transformed linearly over a given number of iterations. This can be used as if a known transformation has been applied to a given data set then if the point cloud application can calculate this transformation in reverse then it can be considered to be accurate and work as intended.

To perform this role the test application was linked to the PCL library this was in order to access methods which would allow for the saving of point clouds to file as PCD files.

The test application can also be used to generate data which fluctuates following the same pattern as a sine curve.

### Experiment Application: Overview

The experiment application was mainly designed to fill the role of producing real world test data to verify the integrity of the point cloud application. The experiment application can fill this role as it was designed to interface with an Arduino microcontroller to manipulate a servo motor at a very particular speed and direction, this speed is matched to the acquisition speed of a Kinect camera. If the Kinect application and point cloud application can be used in conjunction to extract this signal from the servo motor then they can be considered to be accurate and work as intended.

### Process and Methodology

Software development methodologies are patterns which are used to help control the process of developing a large application by using structure and planning. In this instance, the iterative and incremental development methodology was used. This works by setting certain goals which are delegated out in cycles where at the end of each cycle new functionality is added.

To save time and resources as well as to make the program more accessible and understandable self documenting code was used throughout this project as well as including Doxygen comments in order to generate technical documentation.

### Version Control

For this project Git was chosen as the main form of version control, this is because it is the most popular source control method in use at this time. (Ray *et al.*, 2014)

The Git repository for each application was stored on both GitHub and GitLab. (*ALEXJAZZ008008/MSc-Dissertation*, no date) (*ALEXJAZZ008008/KinectLibraries*, no date) (*ALEXJAZZ008008/PointCloudProcessing*, no date) (*ALEXJAZZ008008/KinectPointCloudExperiment*, no date) (*ALEXJAZZ008008/KinectPointCloudTest*, no date) (*Alexander Charles Whitehead / MSc-Dissertation · GitLab*, no date) (*Alexander Charles Whitehead / KinectLibraries · GitLab*, no date) (*Alexander Charles Whitehead / PointCloudProcessing · GitLab*, no date) (*Alexander Charles Whitehead / KinectPointCloudExperiment · GitLab*, no date) (*Alexander Charles Whitehead / KinectPointCloudTest · GitLab*, no date)

# 5 Evaluation

## 5.1 Experiments

### Simple Continuous Plane Test: Overview

The first validation experiment tried was an experiment using test data generated from the test application. This data represented a flat plane at a set distance from the camera. This data was translated one hundred millimetres in every dimension and rotated by one radian in every rotational plane. There were twelve pieces of data in this experiment.

The registration algorithm applied to this data was applied continuously meaning that each point cloud was registered to the point cloud directly before itself. This means that the expected output from this application would be a small linear increase in every rotational plane and transformation dimension, this is because there would be a very small linear change between each subsequent point cloud.

### Simple Continuous Plane Test: Results

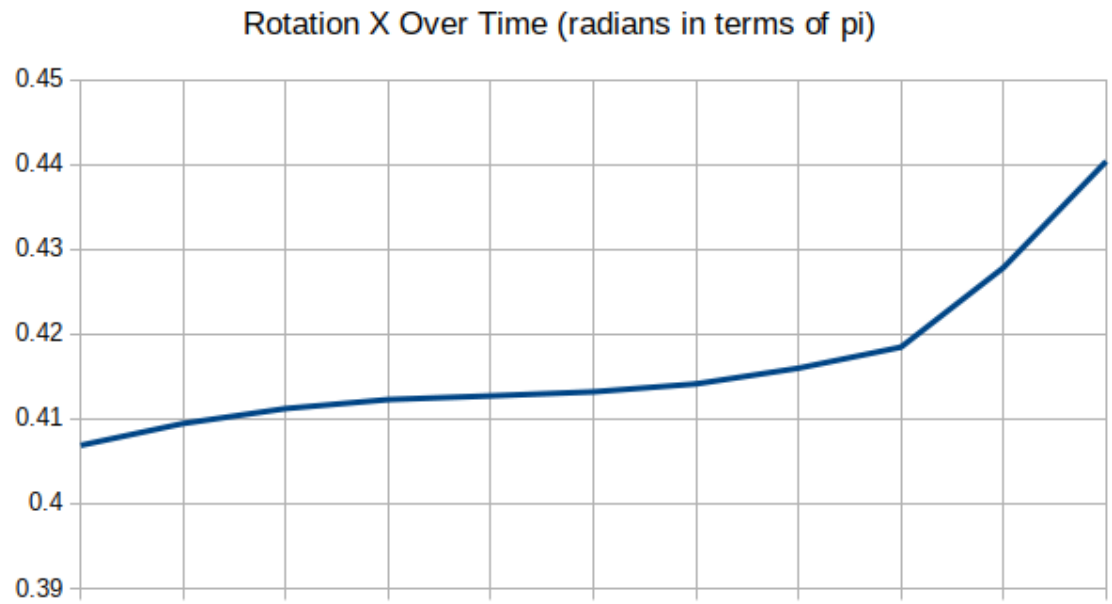


*Illustration 69: This illustration shows the change in rotation in the X plane over time in terms of Pi.*

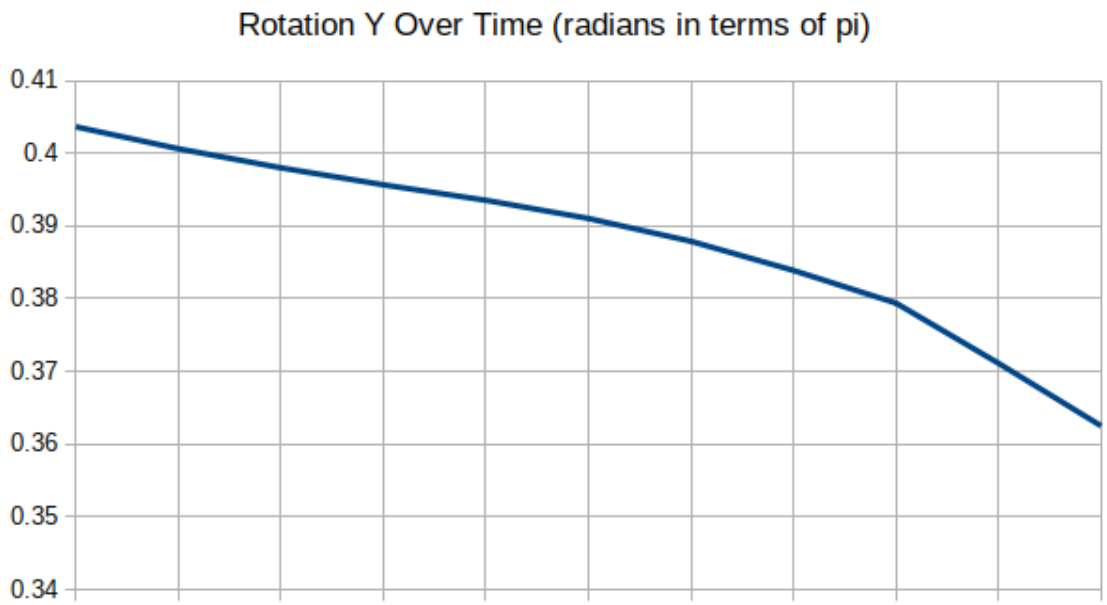


*Illustration 70: This illustration shows the change in rotation in the Y plane over time in terms of Pi.*

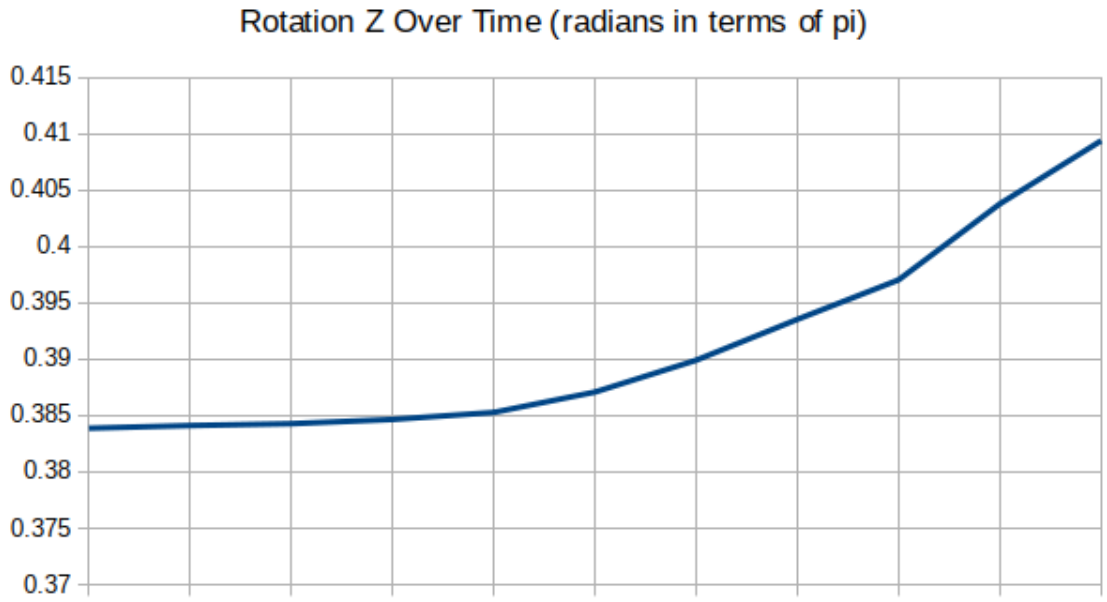


*Illustration 71: This illustration shows the change in rotation in the Z plane over time in terms of Pi.*

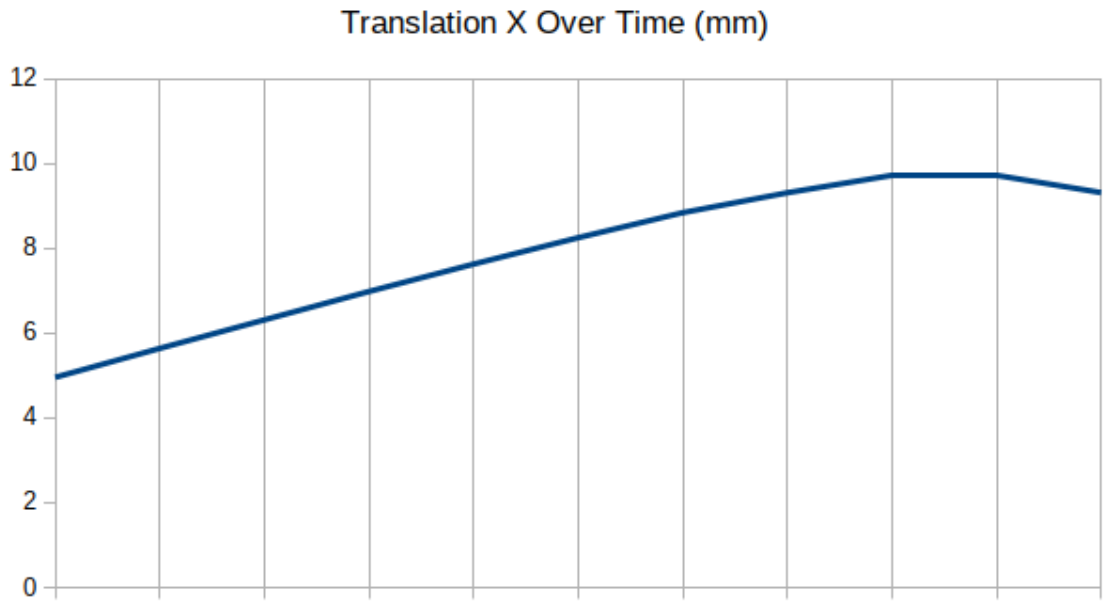


*Illustration 72: This illustration shows the change in translation in the X plane over time in millimetres.*

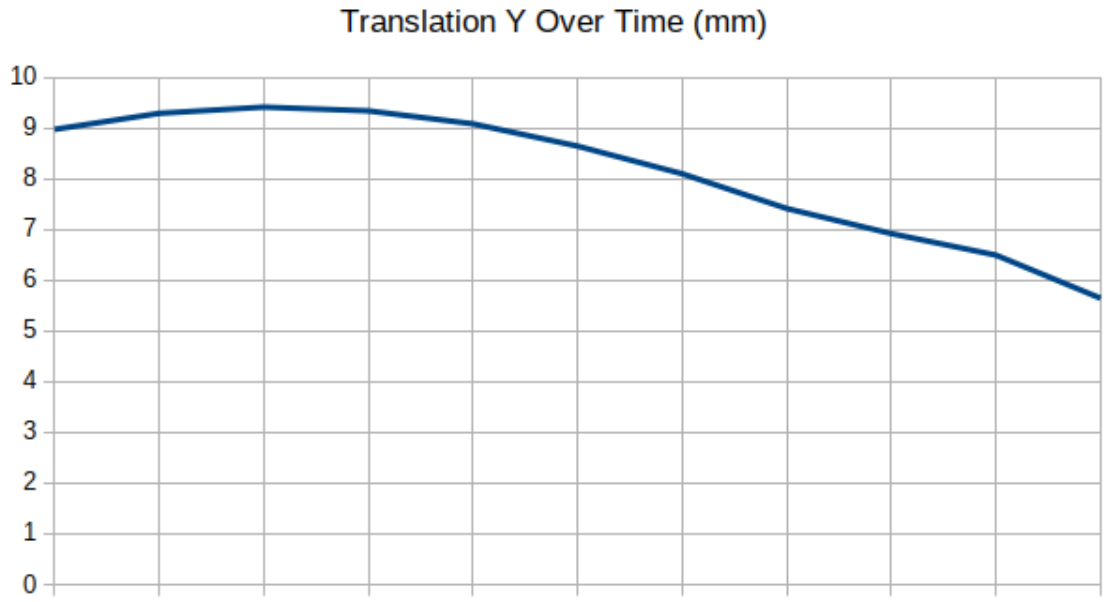


*Illustration 73: This illustration shows the change in translation in the Y plane over time in millimetres.*

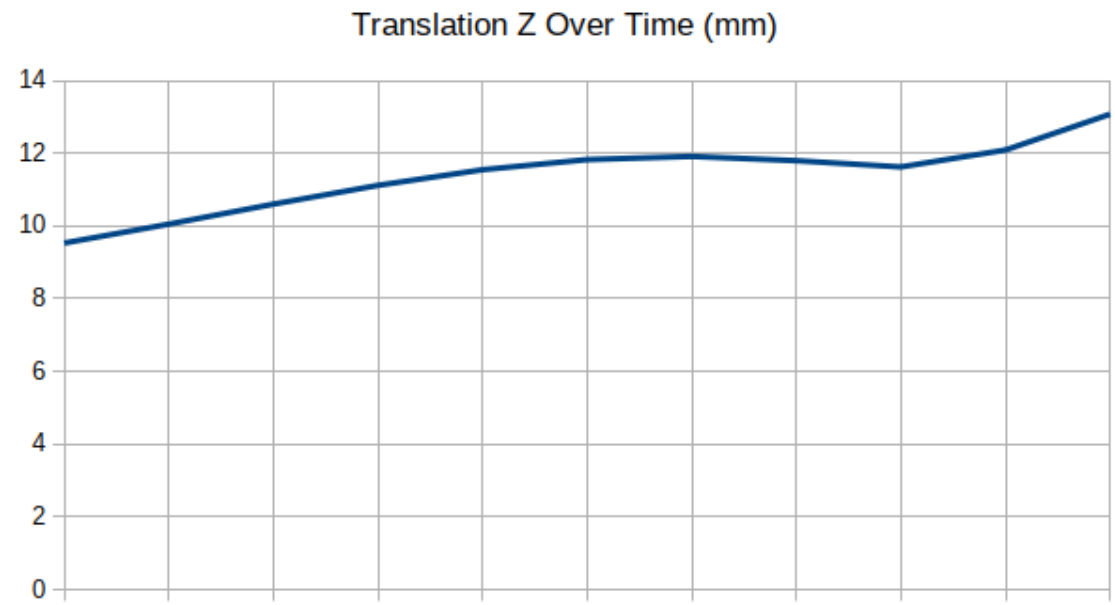


*Illustration 74: This illustration shows the change in translation in the Z plane over time in millimetres.*

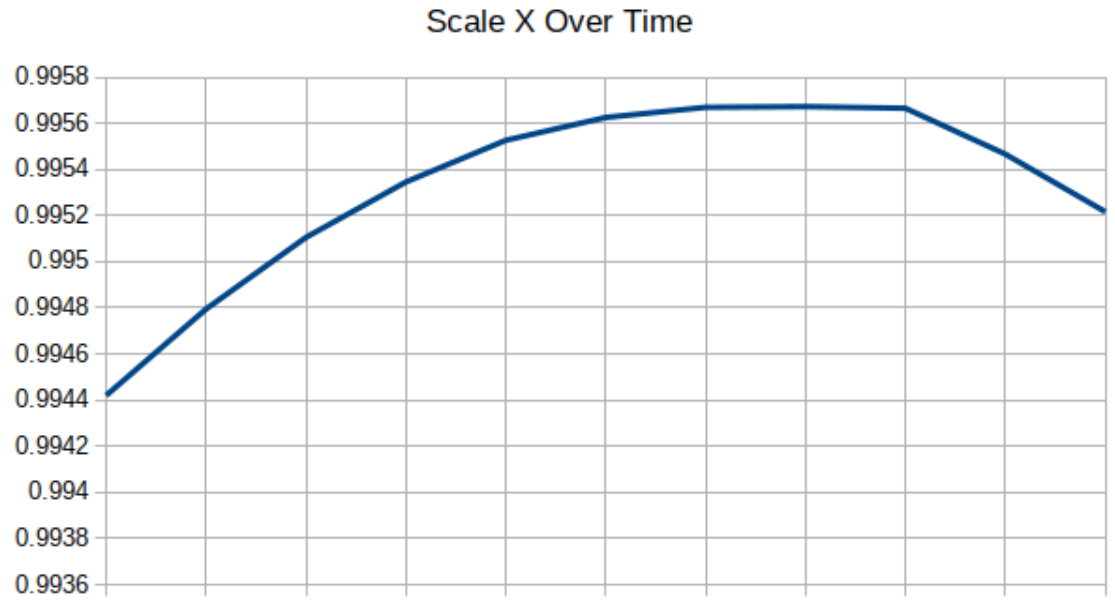


*Illustration 75: This illustration shows the change in scale in the X plane over time in relation to the total size of the object.*

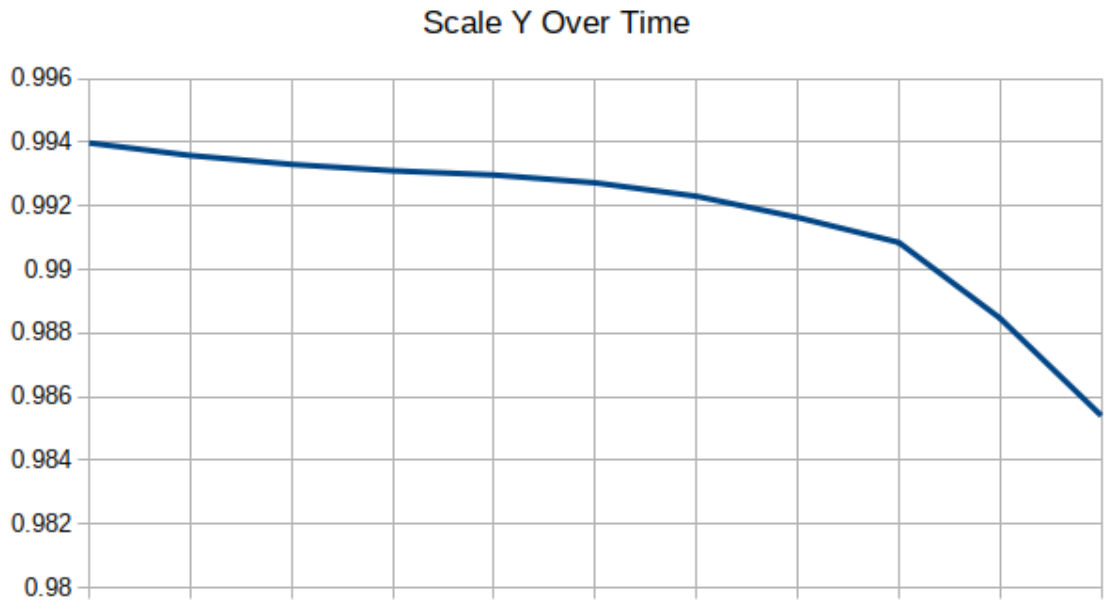


*Illustration 76: This illustration shows the change in scale in the Y plane over time in relation to the total size of the object.*

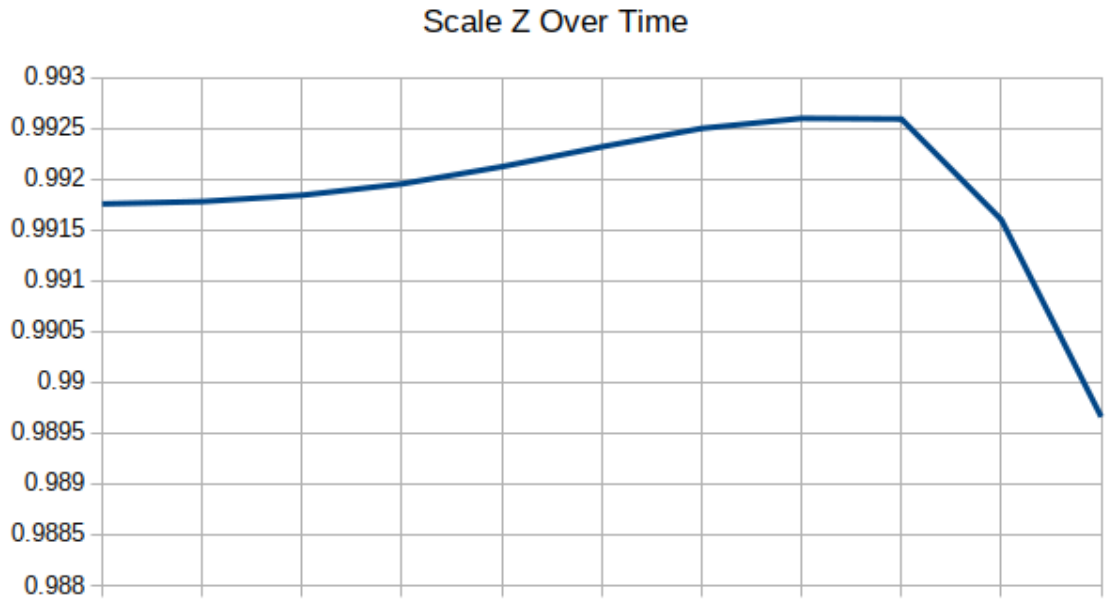


*Illustration 77: This illustration shows the change in scale in the Z plane over time in relation to the total size of the object.*

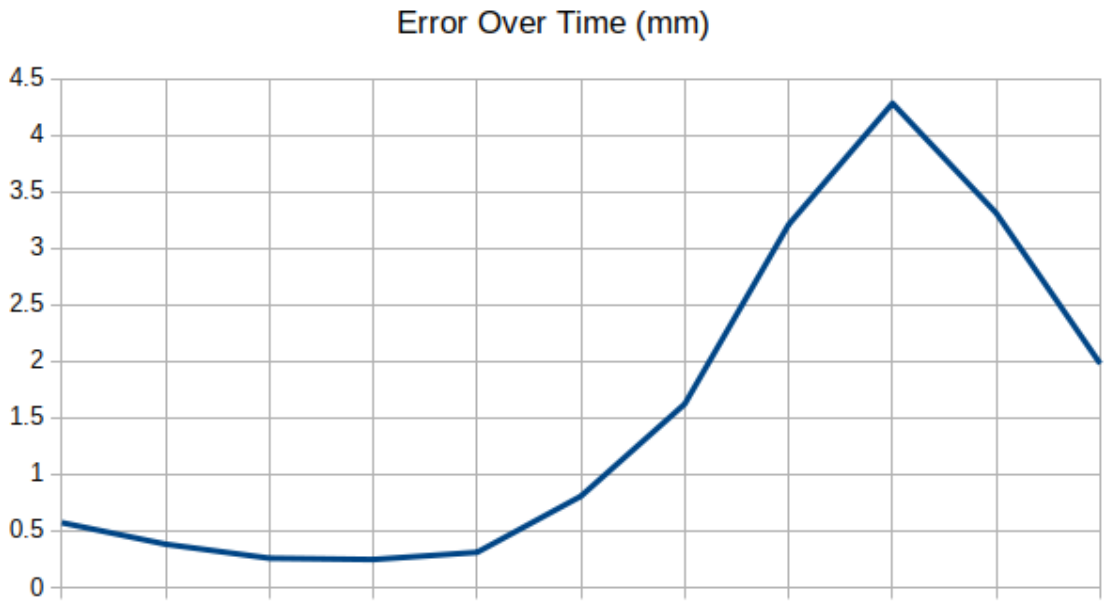


*Illustration 78: This illustration shows the change in error over time in millimetres.*

**Simple Relative Plane Test: Overview**

The next validation experiment tried was an experiment using test data generated from the test application. This data represented a flat plane at a set distance from the camera. This data was translated one hundred millimetres in every dimension and rotated by one radian in every rotational plane. There were twelve pieces of data in this experiment.

The registration algorithm applied to this data was applied relative to the first point cloud meaning that each point cloud was registered to the first point cloud. This experiment is far more accurate to the intended use of the application, than the first experiment, as the output should reflect the cumulative rotation and translation of the motion of the object. This means that the expected output from this application would be a linear increase in every rotational plane and transformation dimension from zero to the values mentioned above.

**Simple Relative Plane Test: Results**

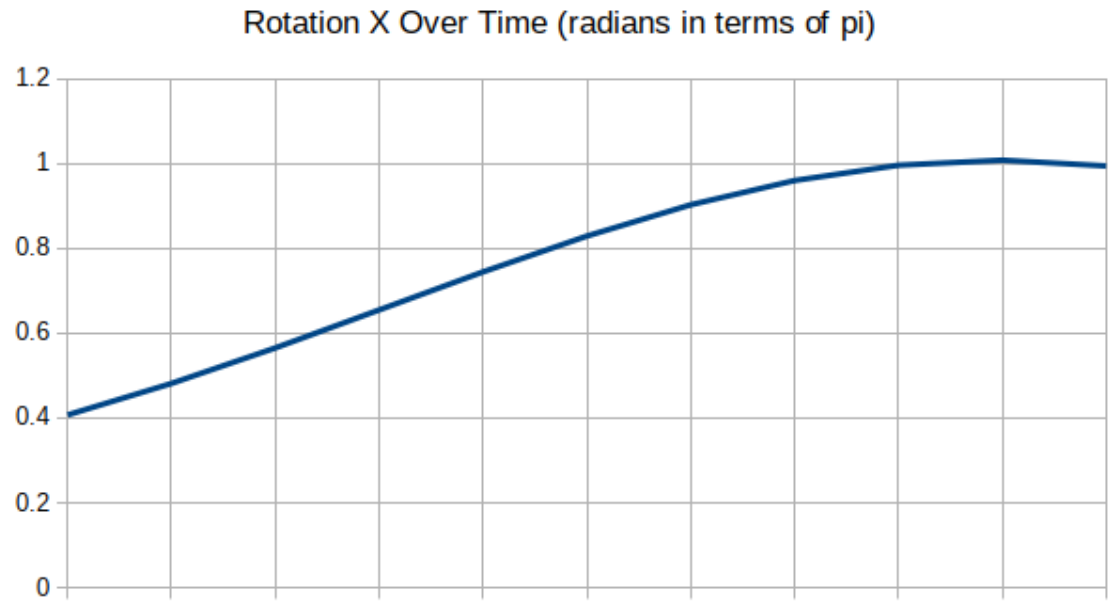


*Illustration 79: This illustration shows the change in rotation in the X plane over time in terms of Pi.*

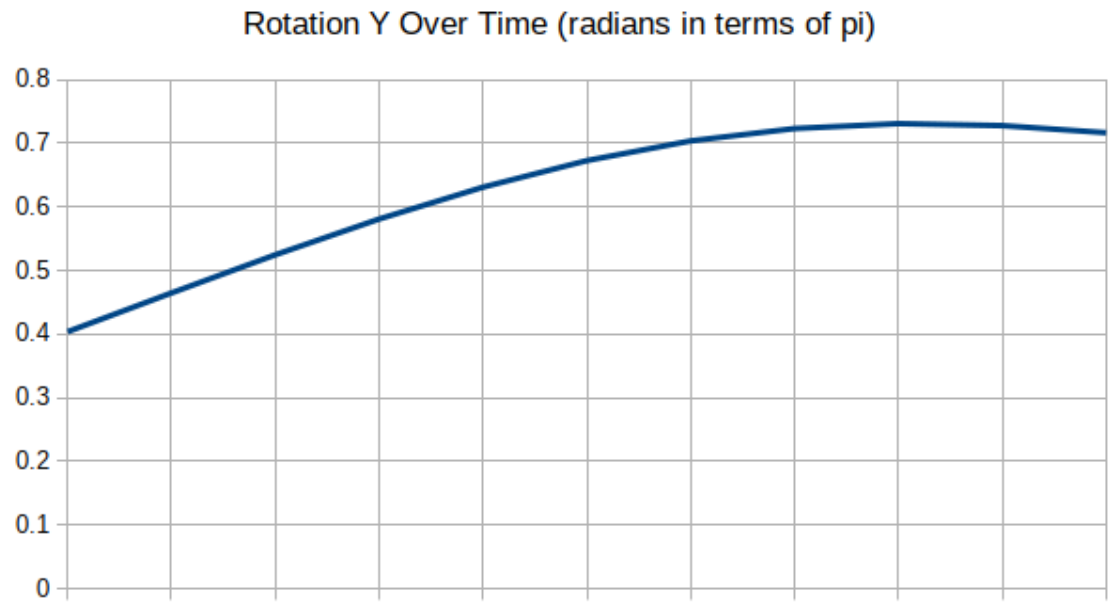


*Illustration 80: This illustration shows the change in rotation in the Y plane over time in terms of Pi.*

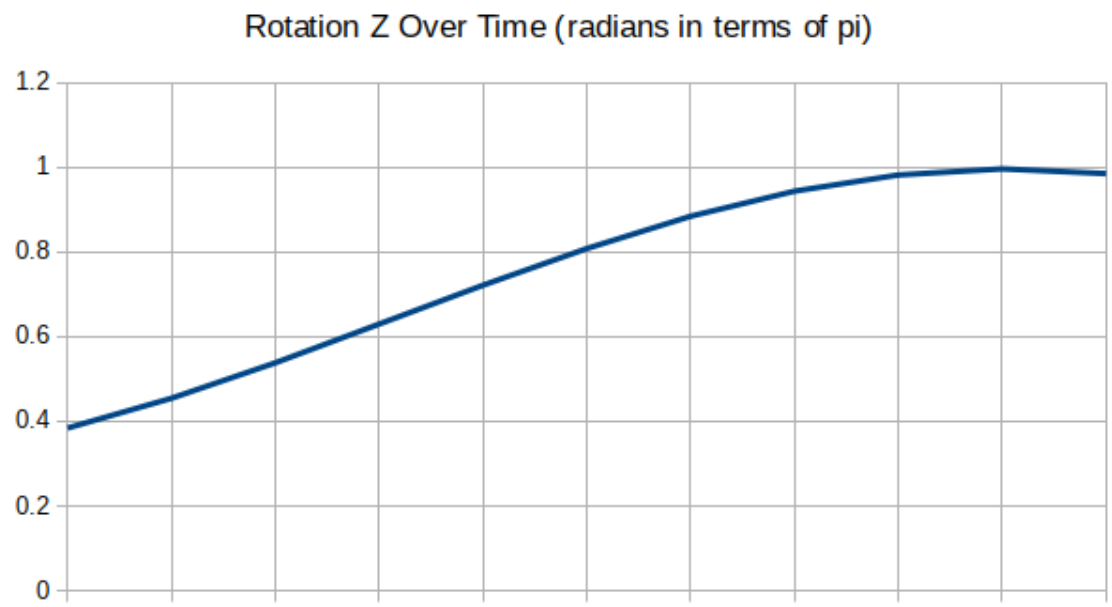


*Illustration 81: This illustration shows the change in rotation in the Z plane over time in terms of Pi.*

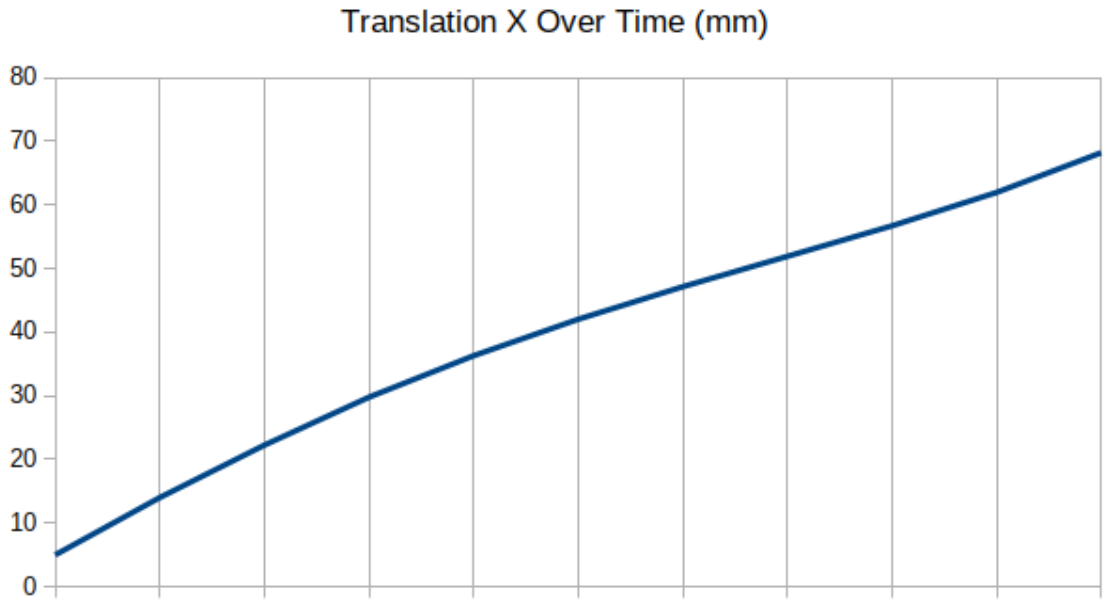


*Illustration 82: This illustration shows the change in translation in the X plane over time in millimetres.*

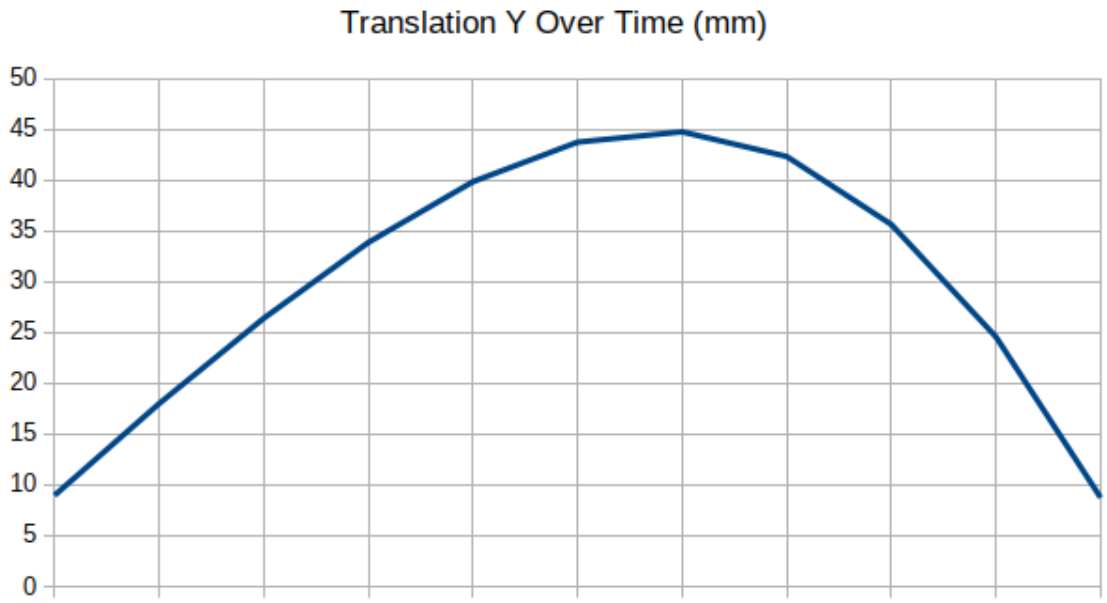


*Illustration 83: This illustration shows the change in translation in the Y plane over time in millimetres.*

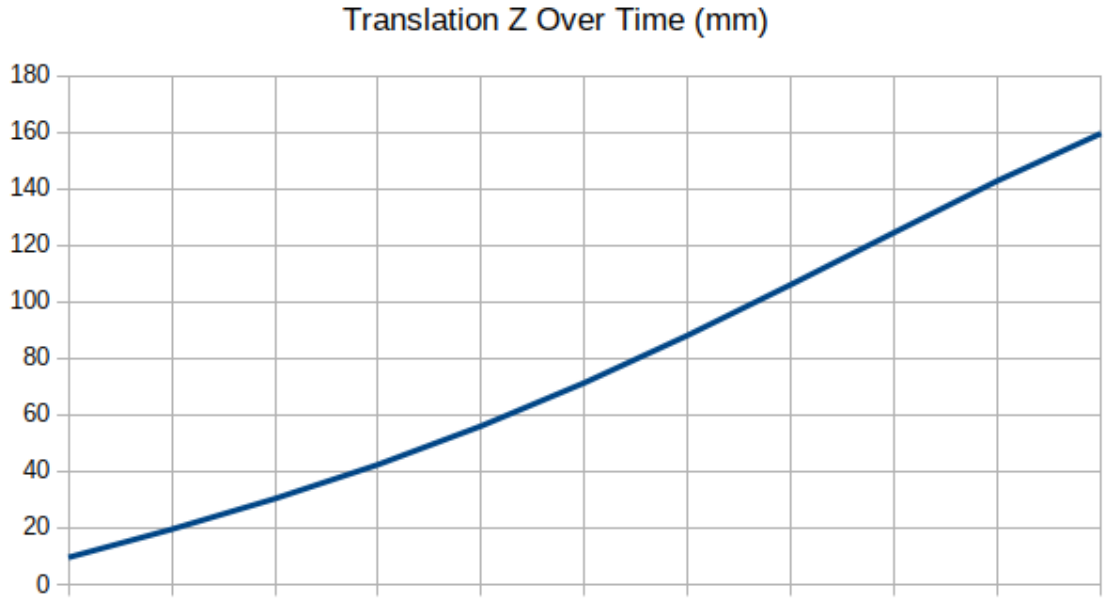


*Illustration 84: This illustration shows the change in translation in the Z plane over time in millimetres.*

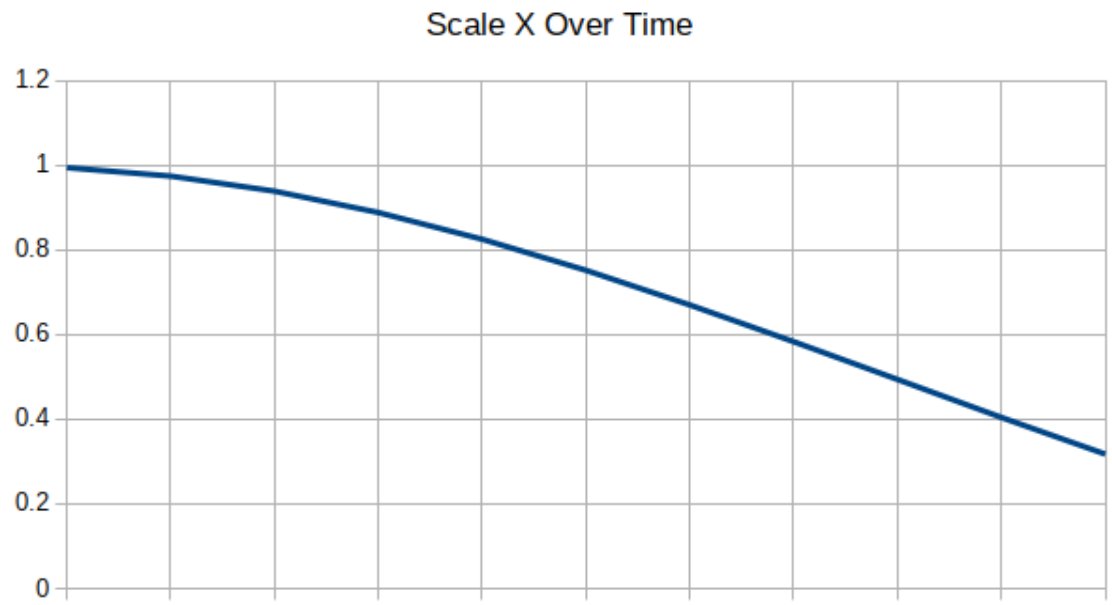


*Illustration 85: This illustration shows the change in scale in the X plane over time in relation to the total size of the object.*

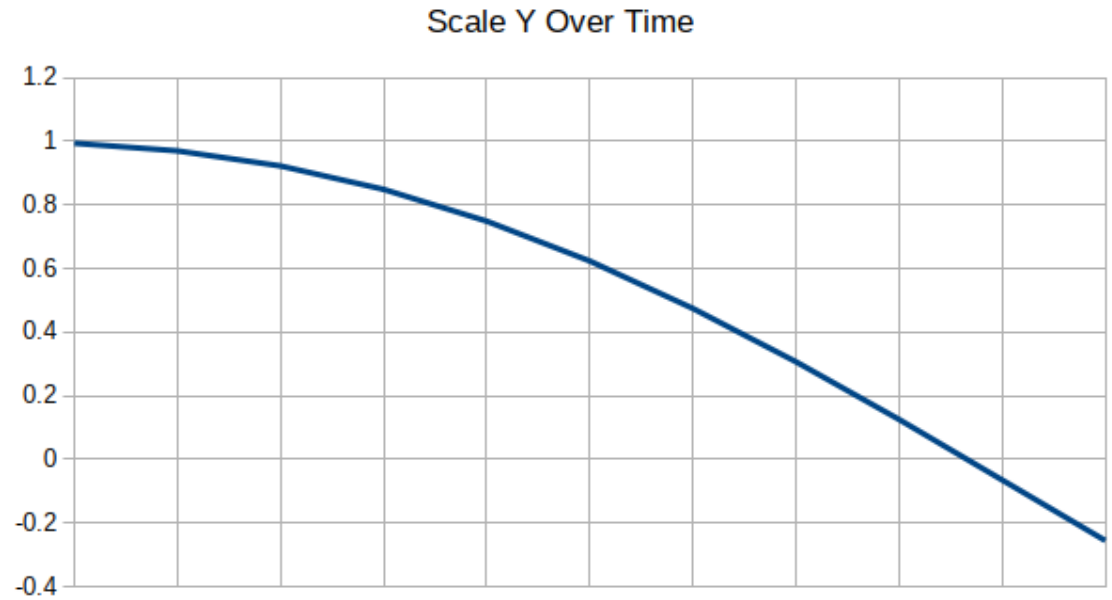


*Illustration 86: This illustration shows the change in scale in the Y plane over time in relation to the total size of the object.*

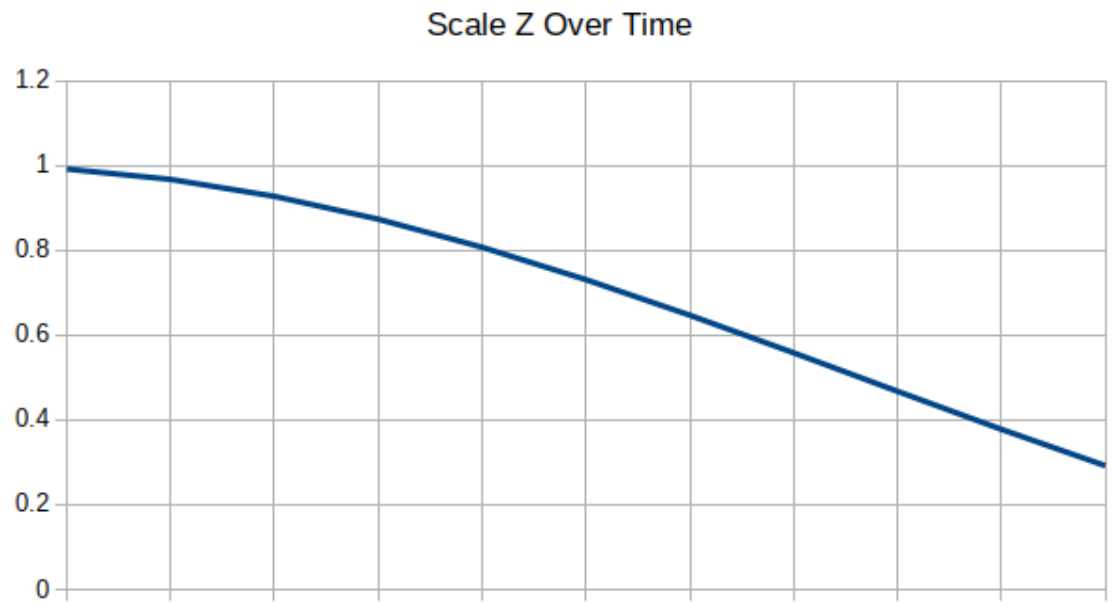


*Illustration 87: This illustration shows the change in scale in the Z plane over time in relation to the total size of the object.*

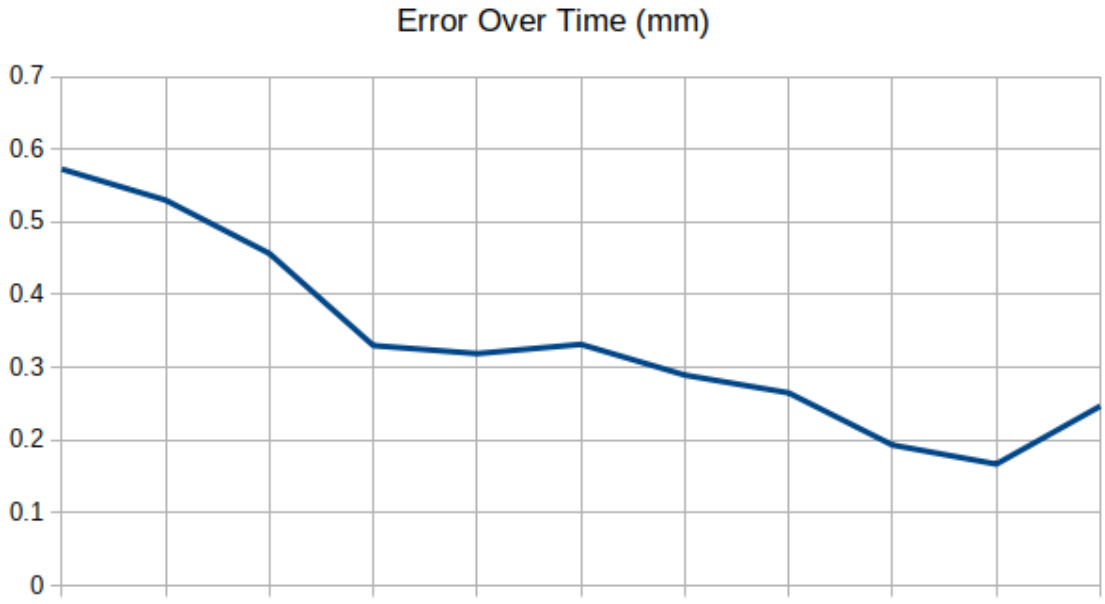


*Illustration 88: This illustration shows the change in error over time in millimetres.*

### Plane Test: Overview

The third validation experiment tried was an experiment using test data generated from the test application. This data represented a flat plane at a set distance from the camera. This data was translated one thousand millimetres in every dimension and rotated by one Pi radian in every rotational plane. There were one thousand and forty pieces of data in this experiment.

The registration algorithm applied to this data was applied relative to the first point cloud meaning that each point cloud was registered to the first point cloud. This experiment is the same as the previous experiment, however the scale of the rotations and translations is much greater and there are far more pieces of data to contend with. This means that the expected output from this application would be a linear increase in every rotational plane up to the centre before a linear decrease in every rotational plane and transformation dimension from zero to the values mentioned above.

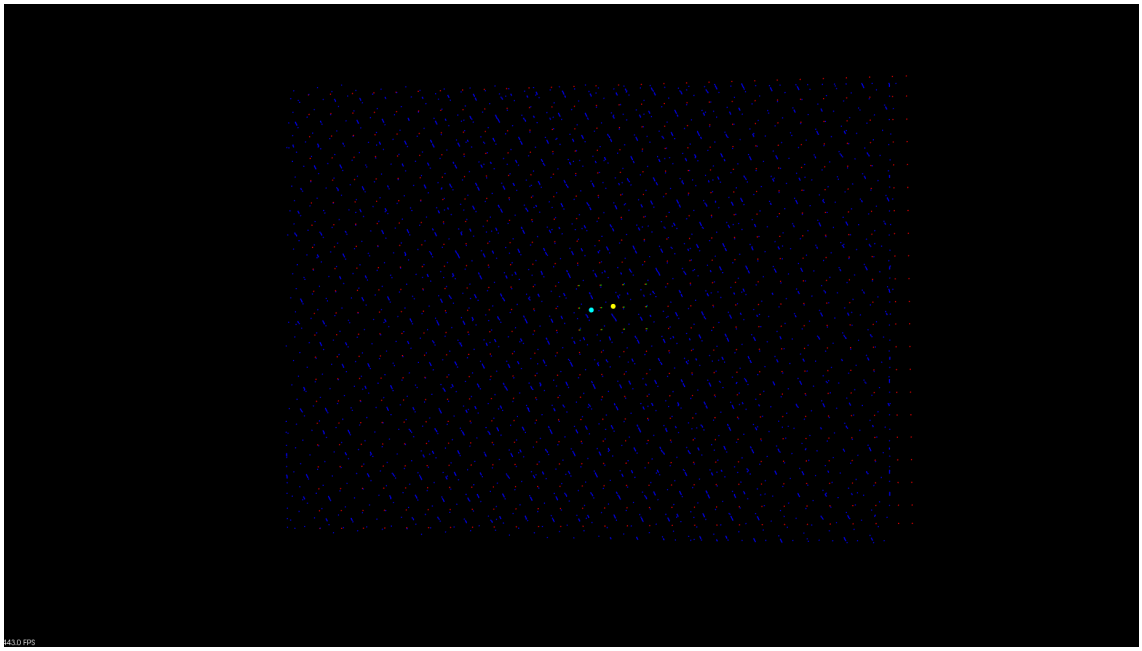

*Illustration 89: This illustration shows the output visualisation of the final pair of point clouds from the front.*

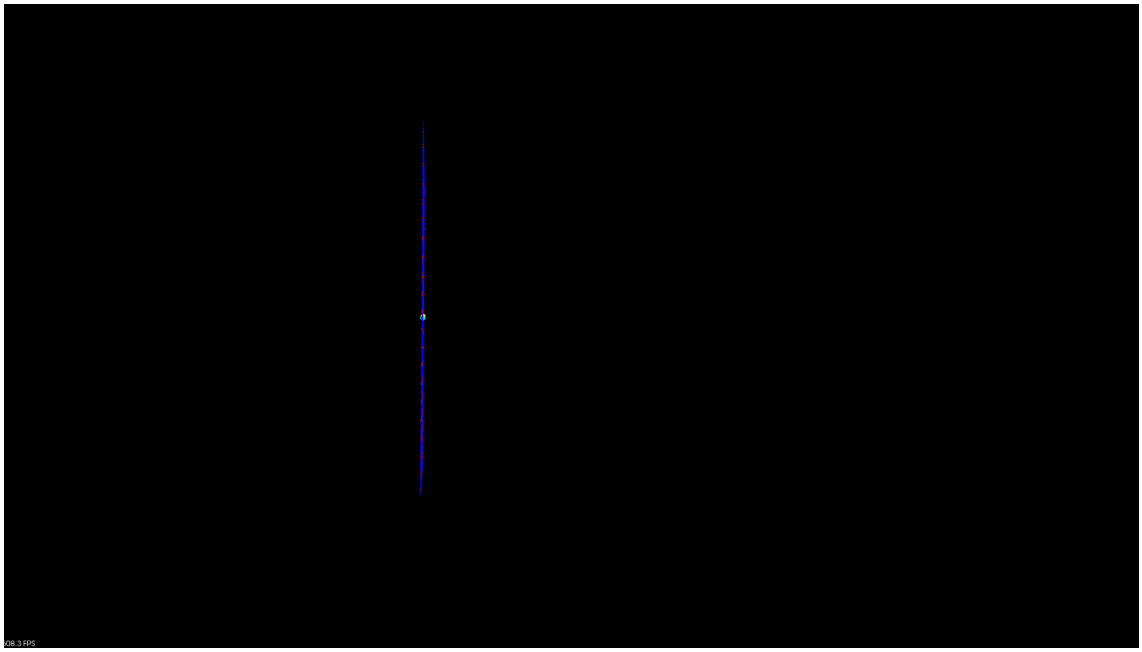

*Illustration 90: This illustration shows the output visualisation of the final pair of point clouds from the side.*

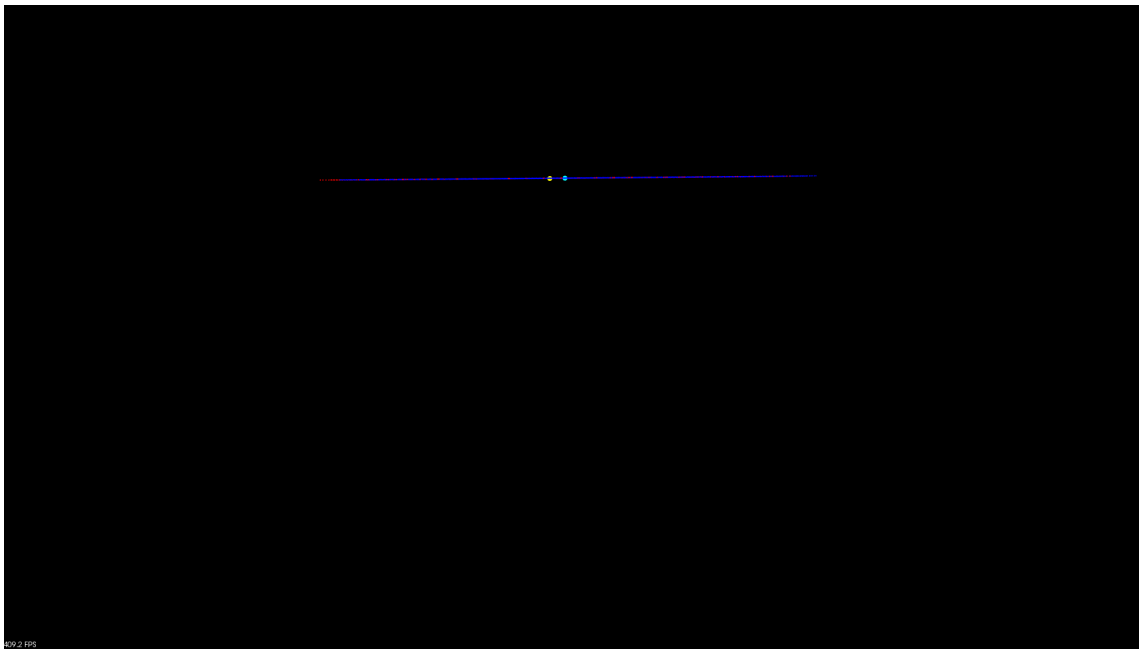

*Illustration 91: This illustration shows the output visualisation of the final pair of point clouds from the top.*

**Plane Test: Results**

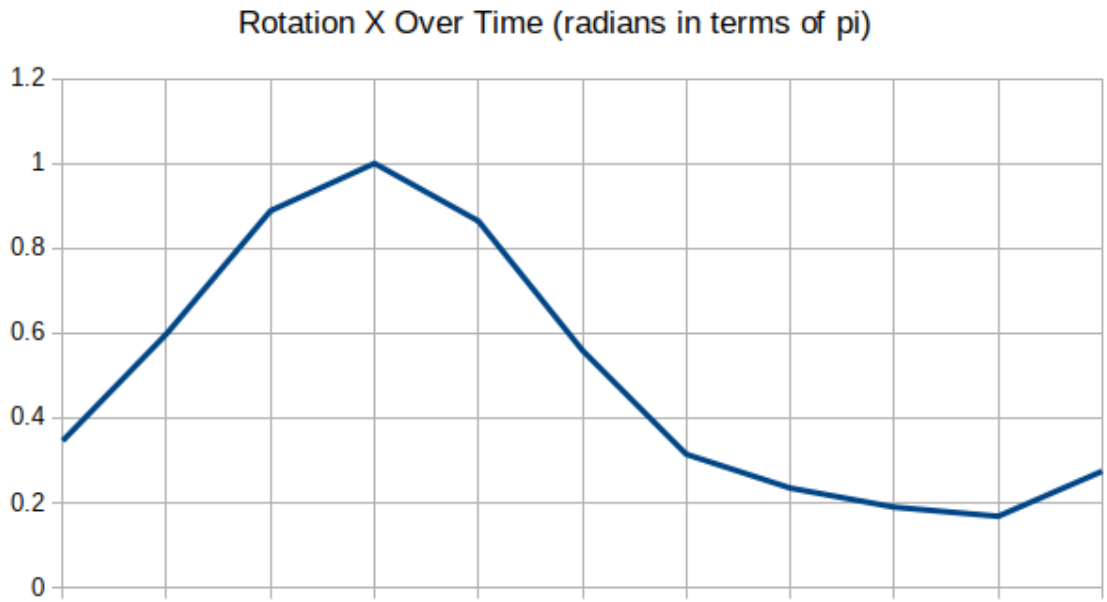


*Illustration 92: This illustration shows the change in rotation in the X plane over time in terms of Pi.*

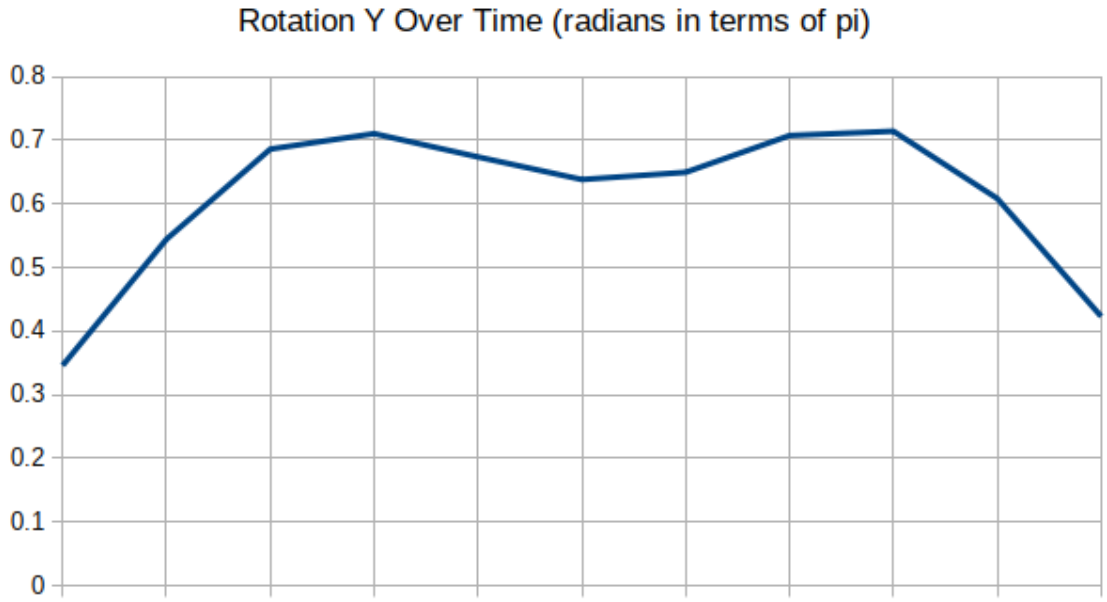


*Illustration 93: This illustration shows the change in rotation in the Y plane over time in terms of Pi.*

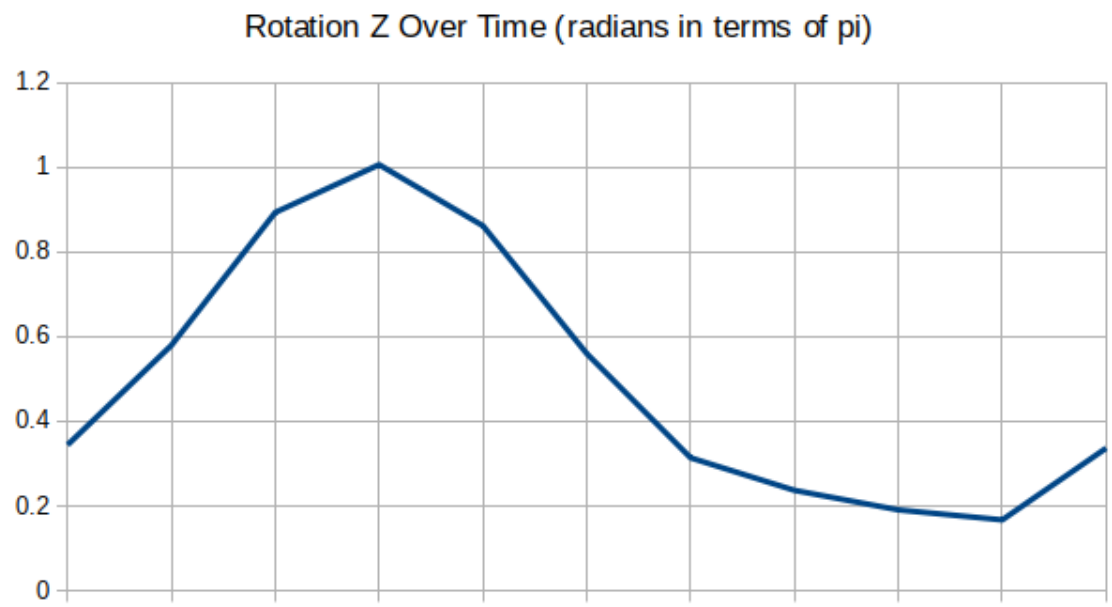


*Illustration 94: This illustration shows the change in rotation in the Z plane over time in terms of Pi.*

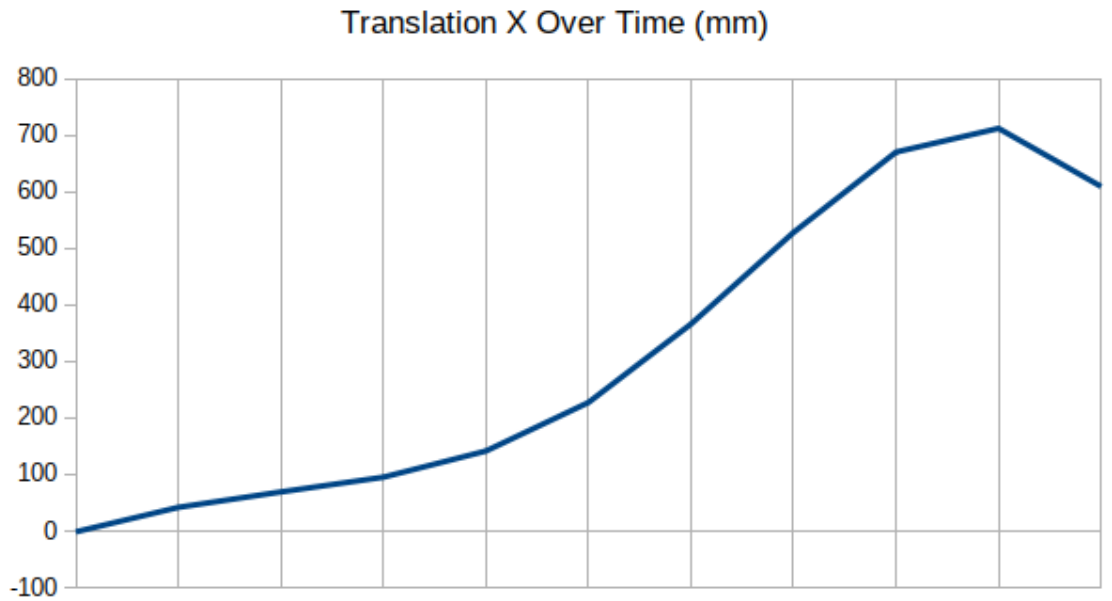


*Illustration 95: This illustration shows the change in translation in the X plane over time in millimetres.*

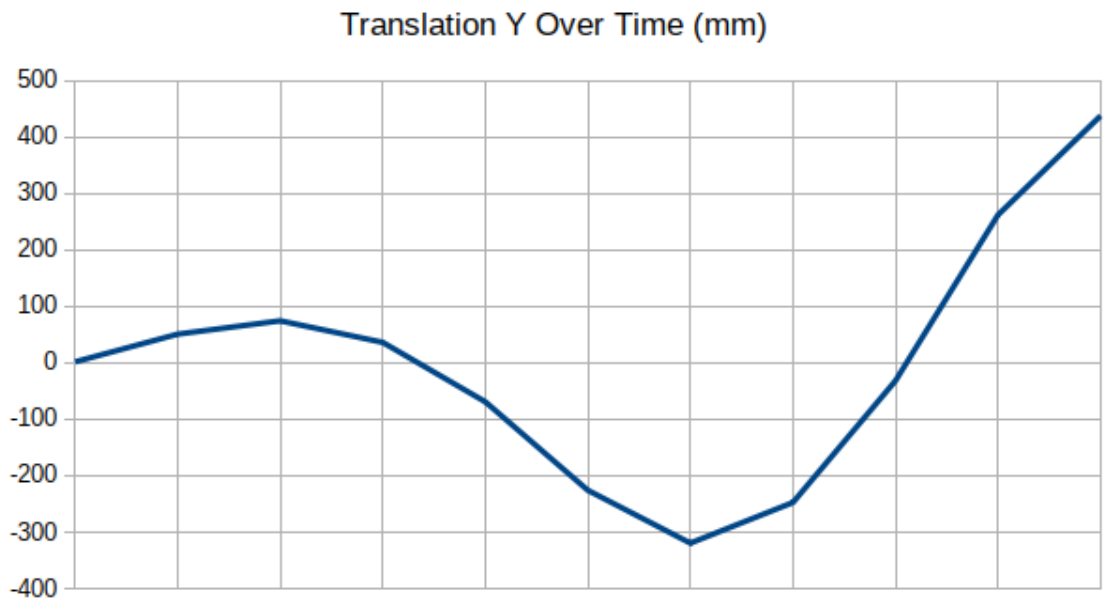


*Illustration 96: This illustration shows the change in translation in the Y plane over time in millimetres.*

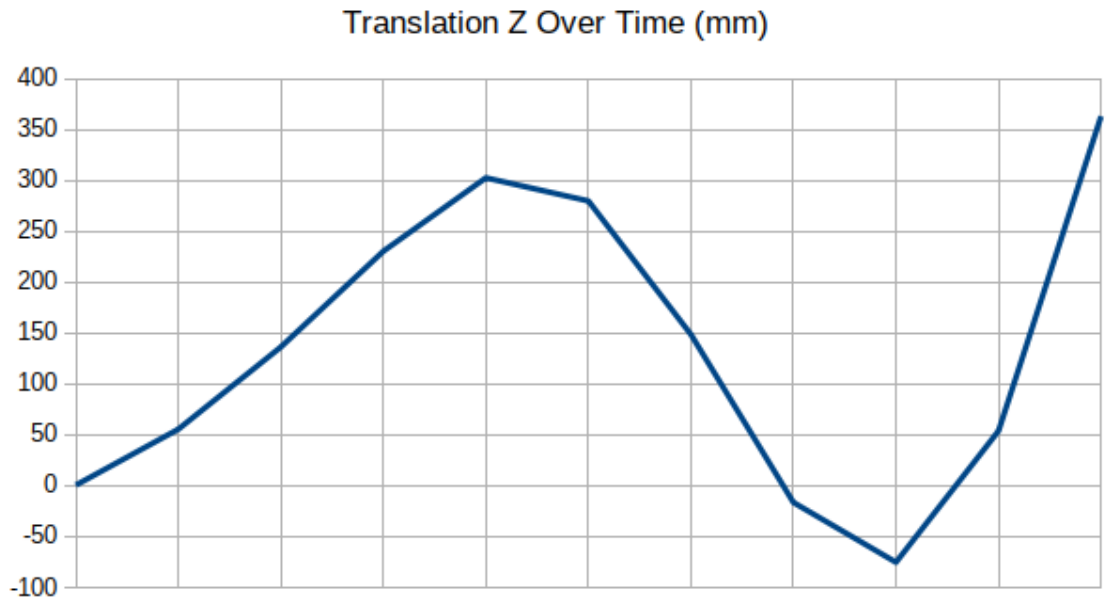


*Illustration 97: This illustration shows the change in translation in the Z plane over time in millimetres.*

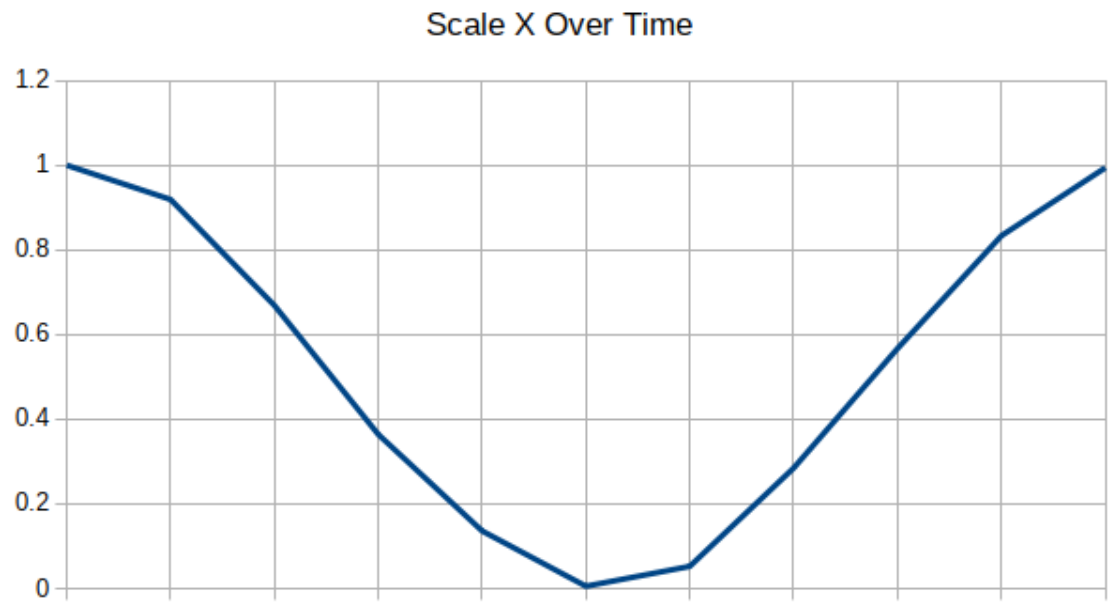


*Illustration 98: This illustration shows the change in scale in the X plane over time in relation to the total size of the object.*

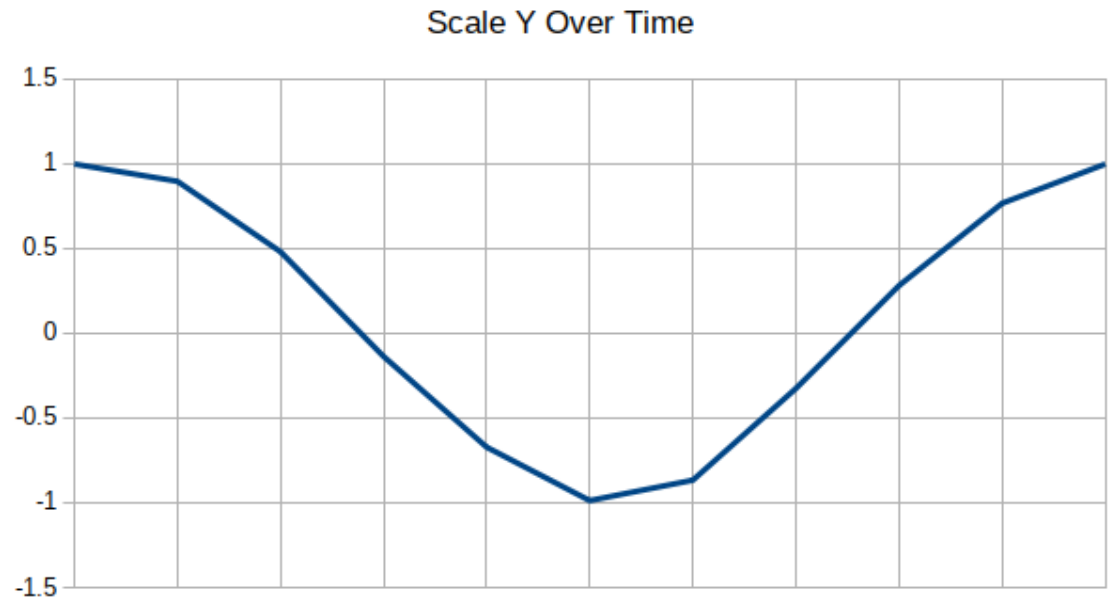


*Illustration 99: This illustration shows the change in scale in the Y plane over time in relation to the total size of the object.*

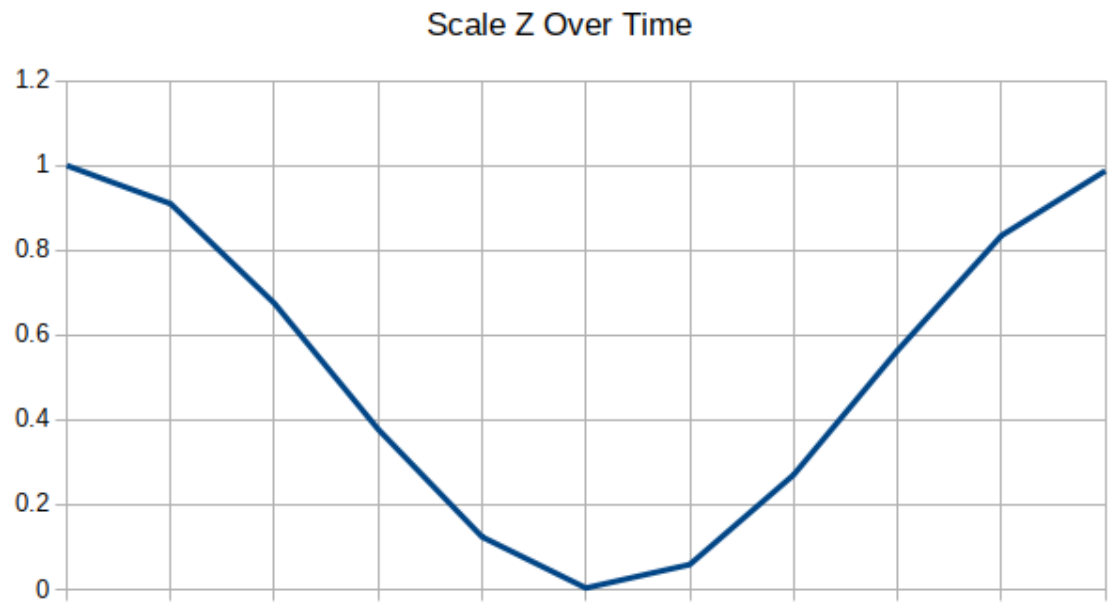


*Illustration 100: This illustration shows the change in scale in the Z plane over time in relation to the total size of the object.*

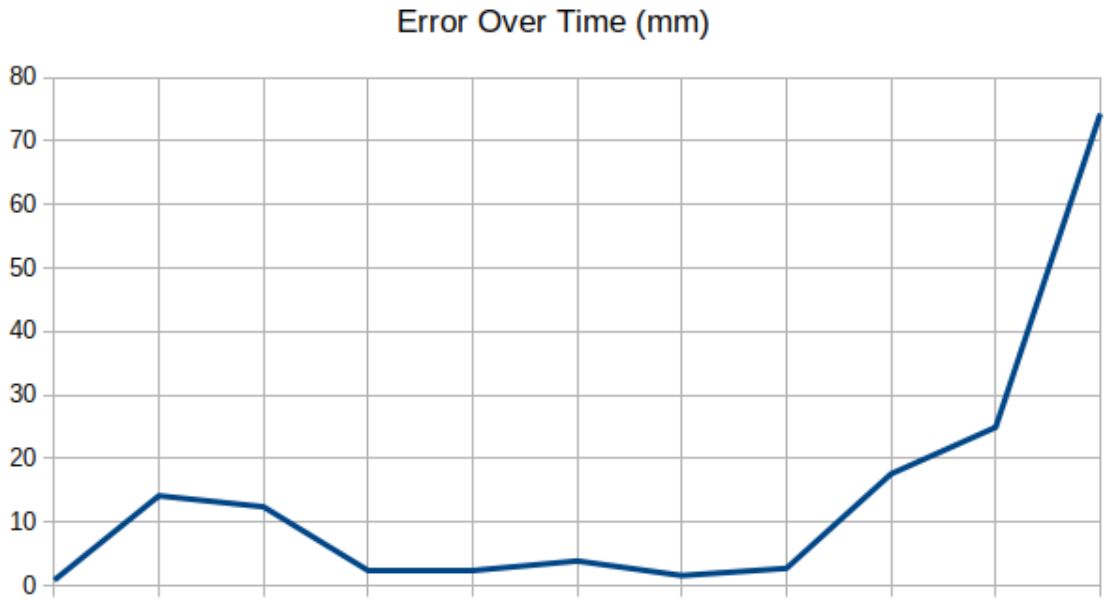


*Illustration 101: This illustration shows the change in error over time in millimetres.*

### Curve Test: Overview

The fourth validation experiment tried was an experiment using test data generated from the test application. This data represented a curved surface generated from a collection of sine curves at a set distance from the camera. This data was translated one thousand millimetres in every dimension and rotated by one Pi radian in every rotational plane. There were one thousand and forty pieces of data in this experiment.

The registration algorithm applied to this data was applied relative to the first point cloud meaning that each point cloud was registered to the first point cloud. This experiment is the same as the previous experiment, however a curved surface was used rather than a flat plane. This means that the expected output from this application would be a linear increase in every rotational plane up to the centre before a linear decrease in every rotational plane and transformation dimension from zero to the values mentioned above.

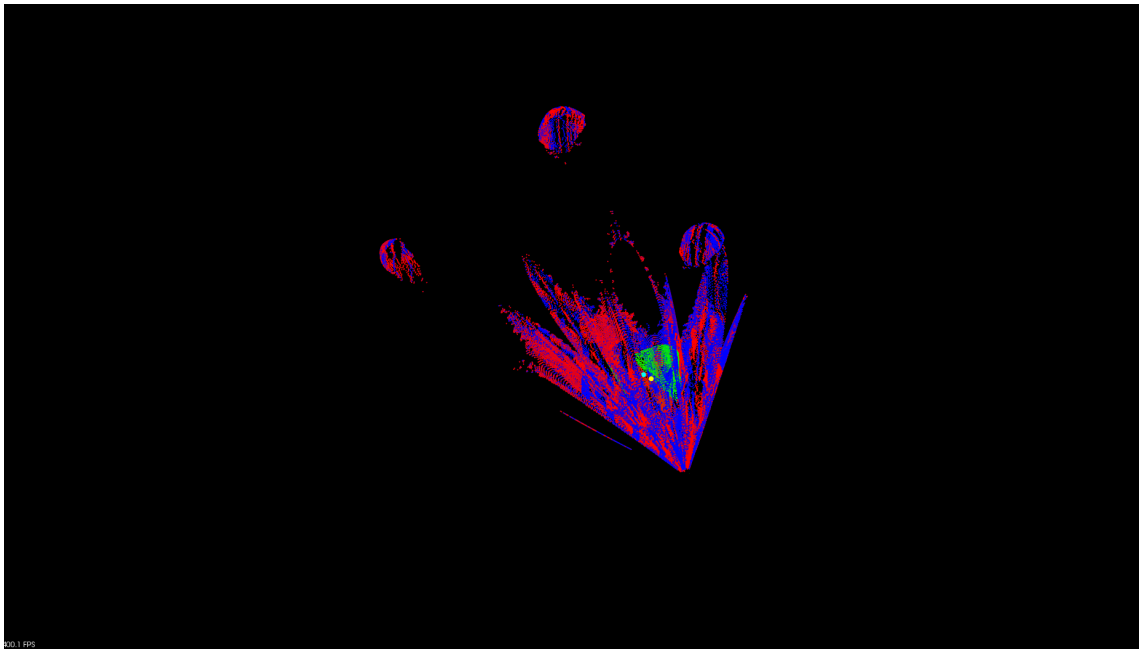

*Illustration 102: This illustration shows the output visualisation of the final pair of point clouds from the front.*

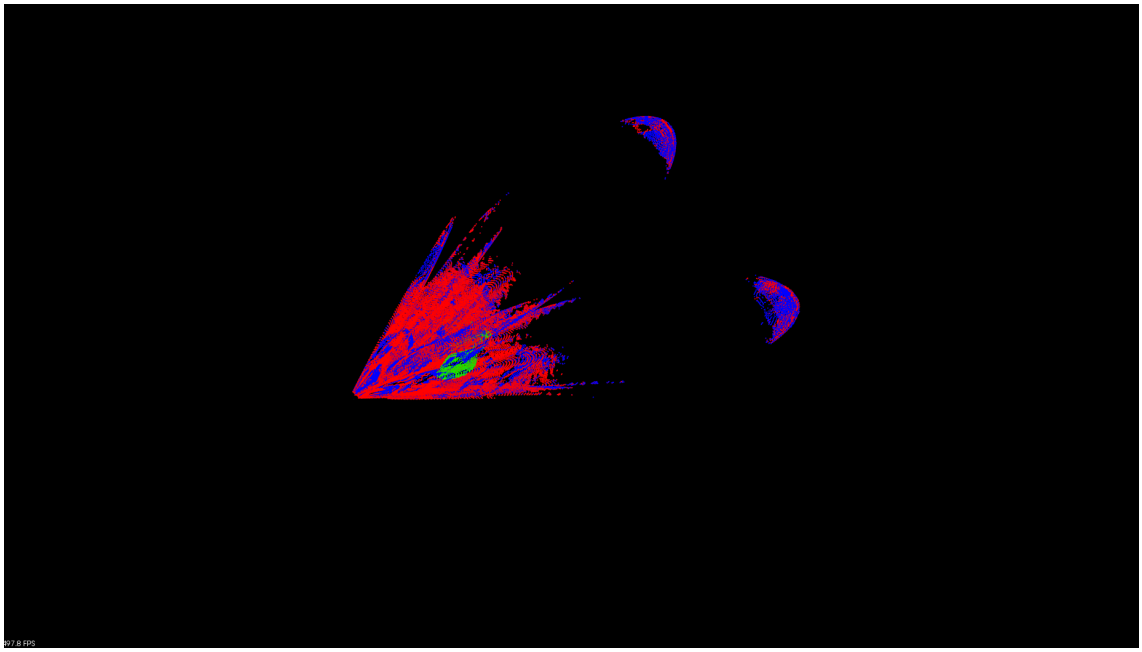

*Illustration 103: This illustration shows the output visualisation of the final pair of point clouds from the side.*

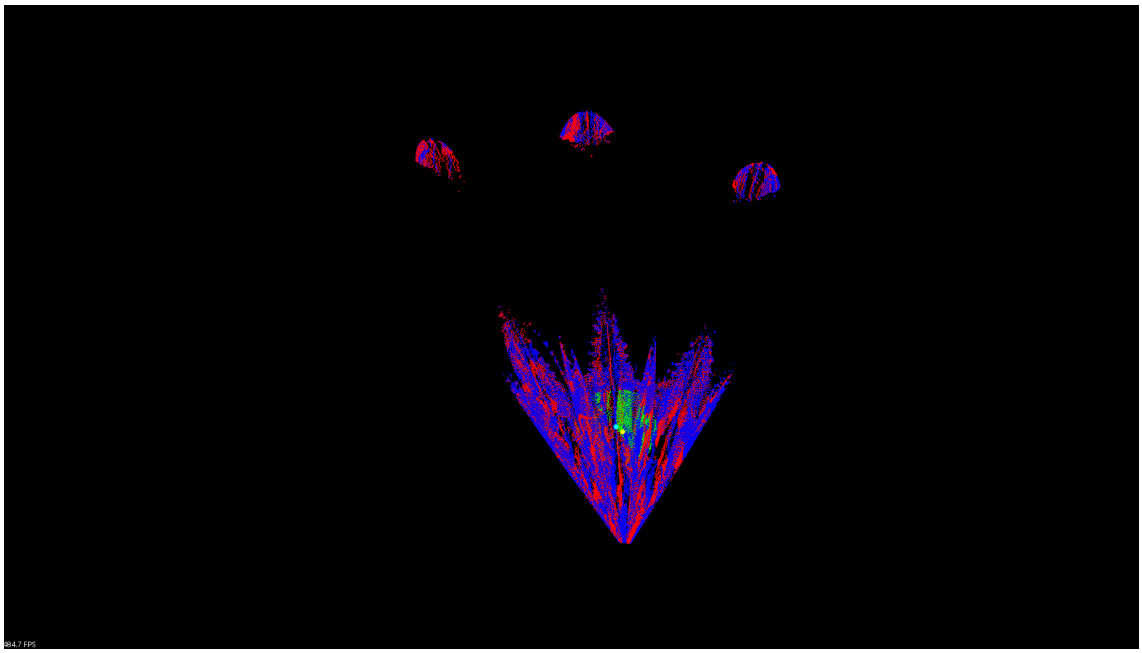

*Illustration 104: This illustration shows the output visualisation of the final pair of point clouds from the top.*

**Curve Test: Results**

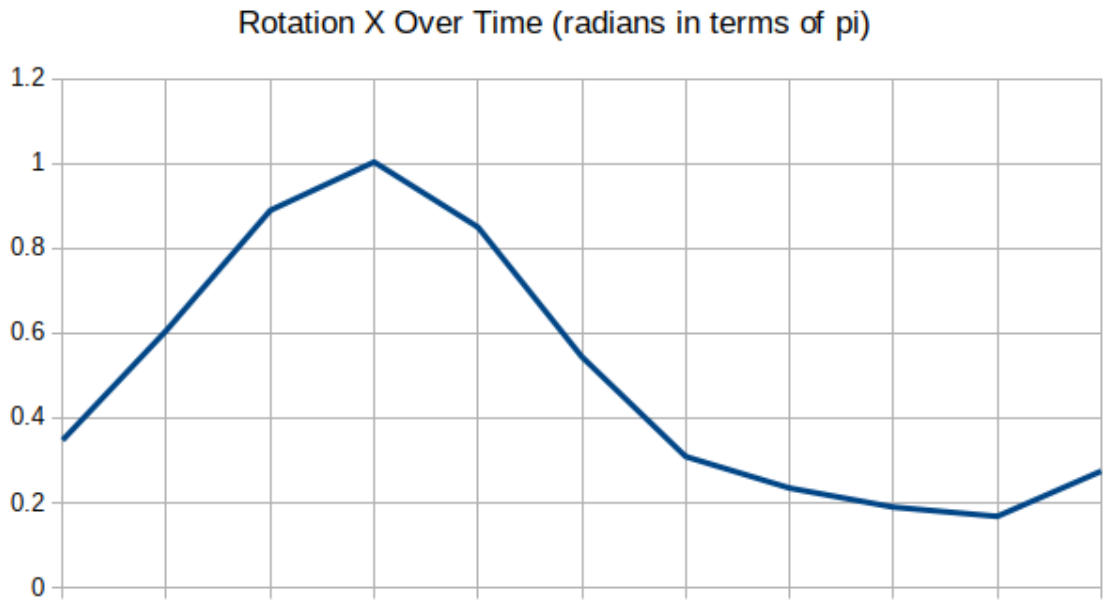


*Illustration 105: This illustration shows the change in rotation in the X plane over time in terms of Pi.*

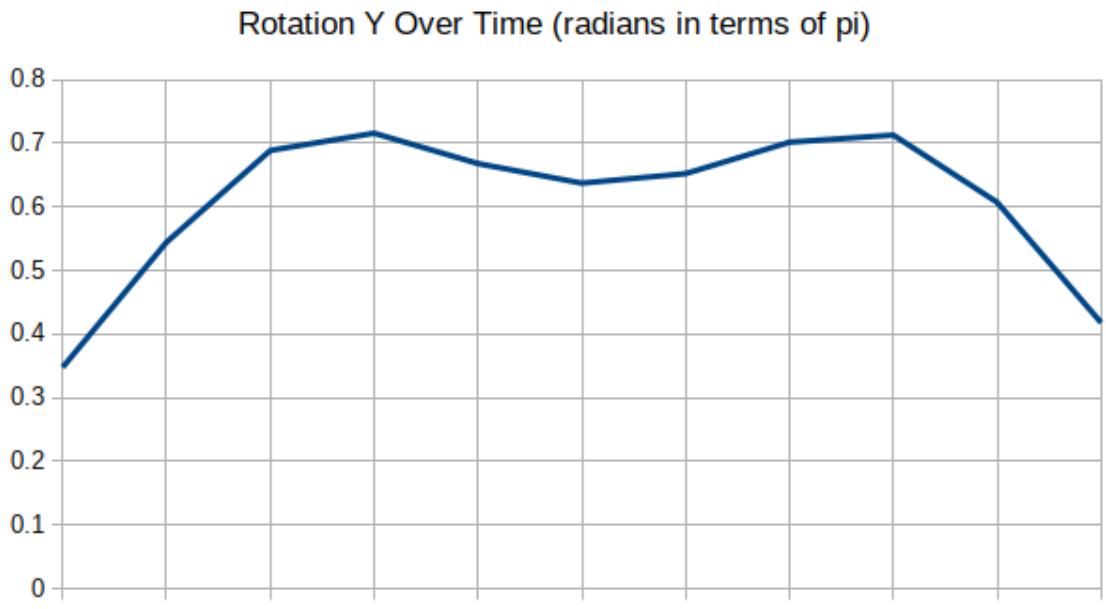


*Illustration 106: This illustration shows the change in rotation in the Y plane over time in terms of Pi.*

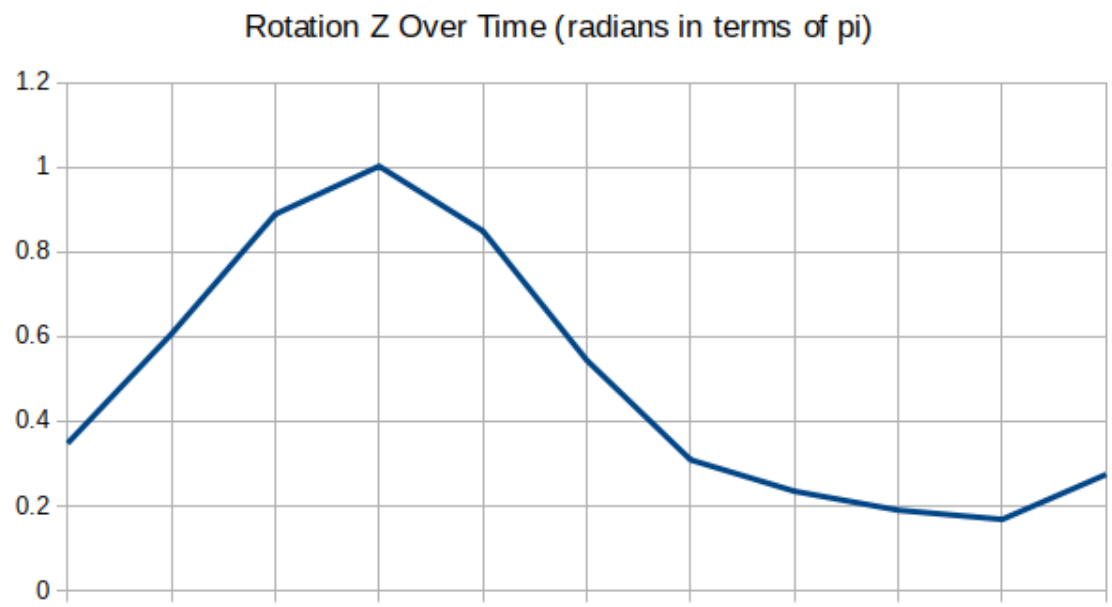


*Illustration 107: This illustration shows the change in rotation in the Z plane over time in terms of Pi.*

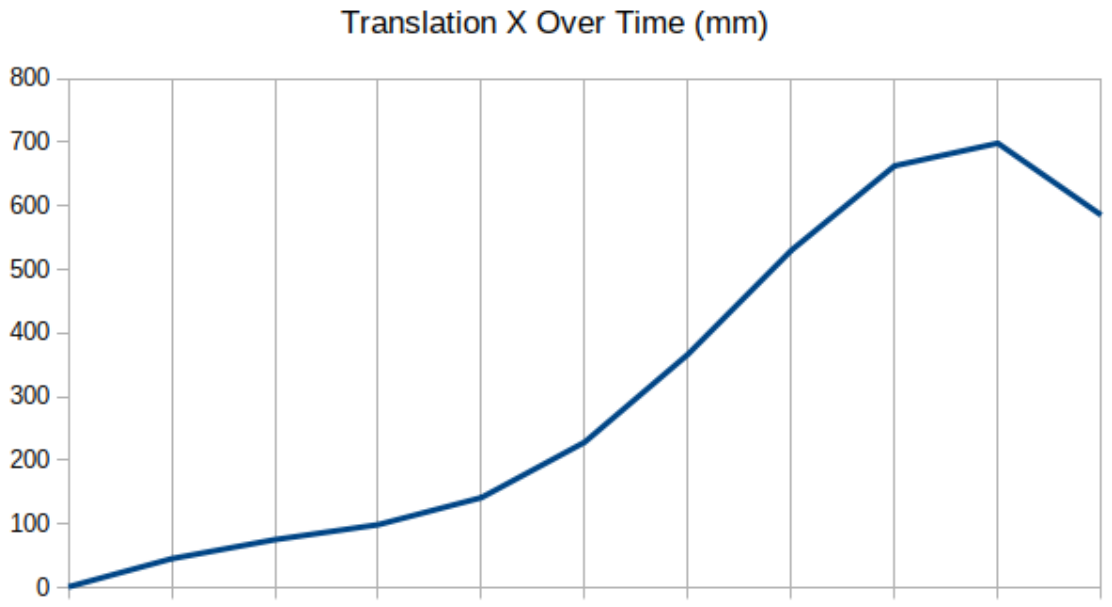


*Illustration 108: This illustration shows the change in translation in the X plane over time in millimetres.*

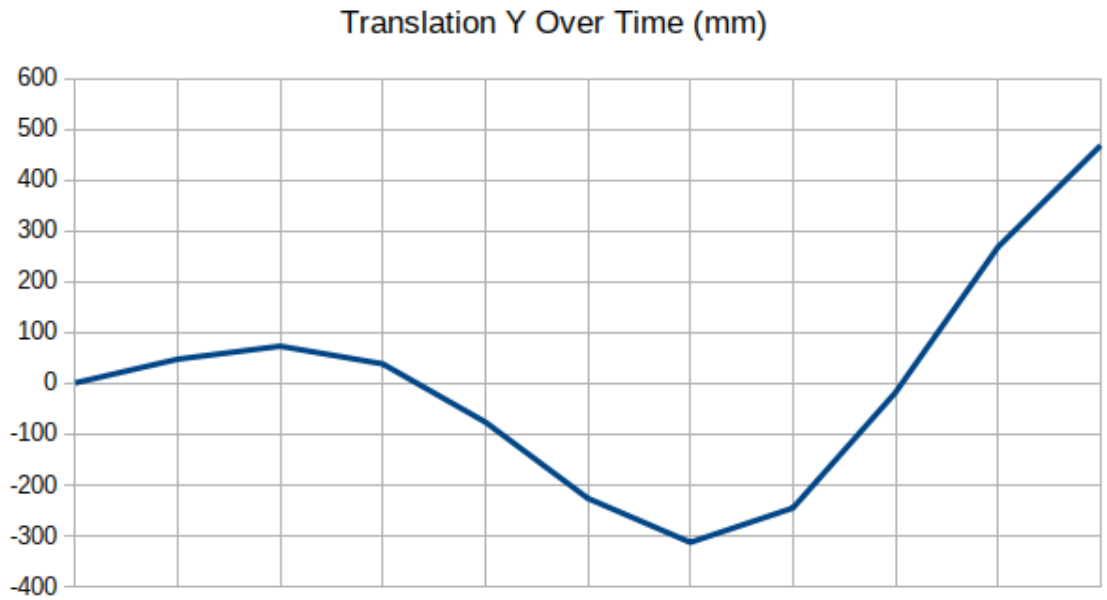


*Illustration 109: This illustration shows the change in translation in the Y plane over time in millimetres.*

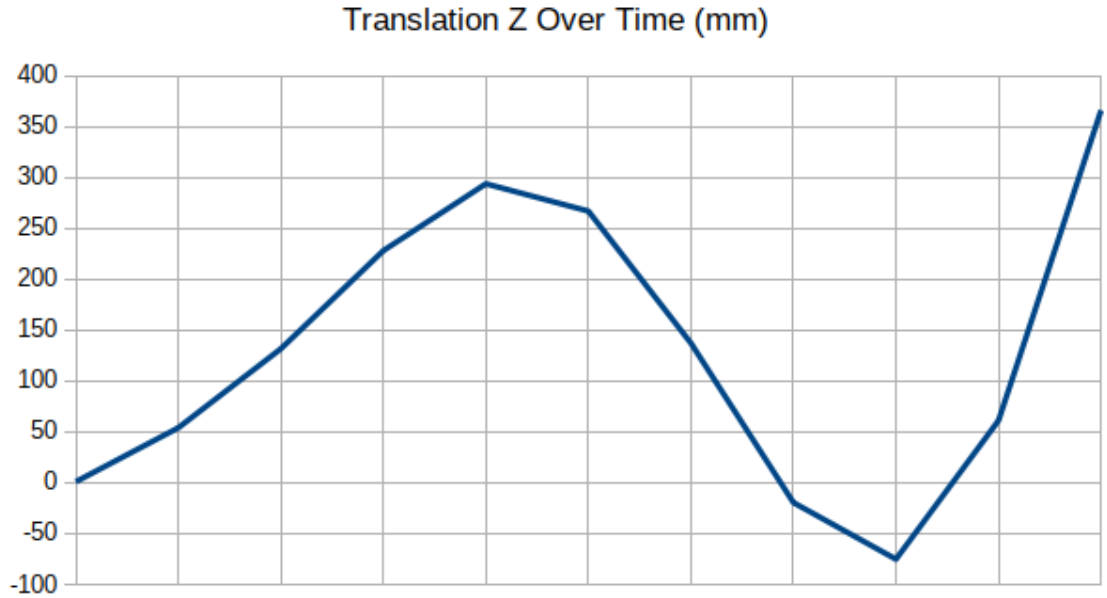


*Illustration 110: This illustration shows the change in translation in the Z plane over time in millimetres.*

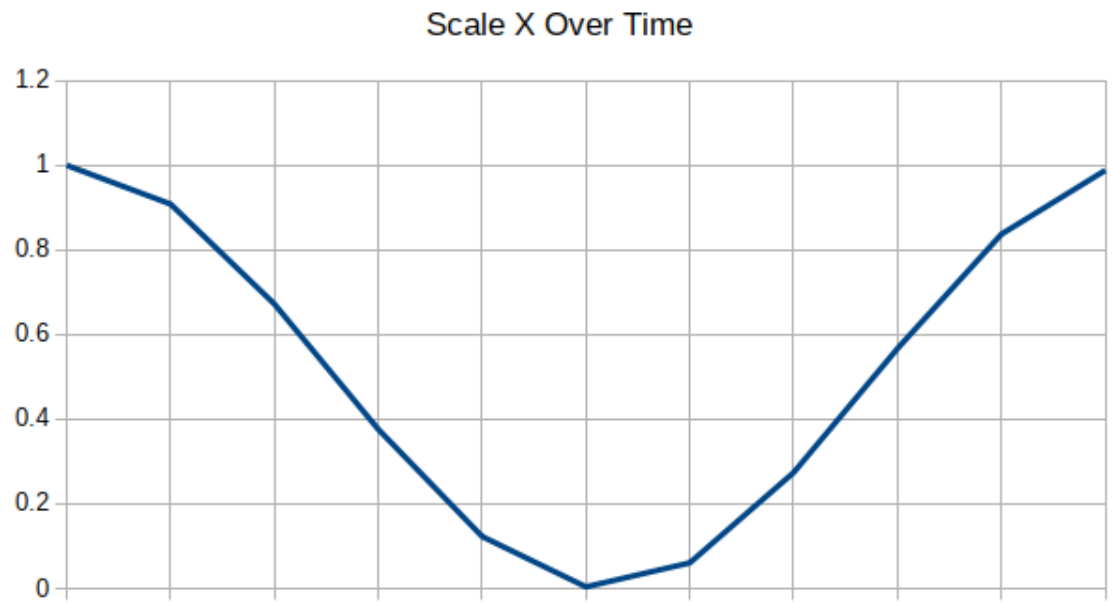


*Illustration 111: This illustration shows the change in scale in the X plane over time in relation to the total size of the object.*

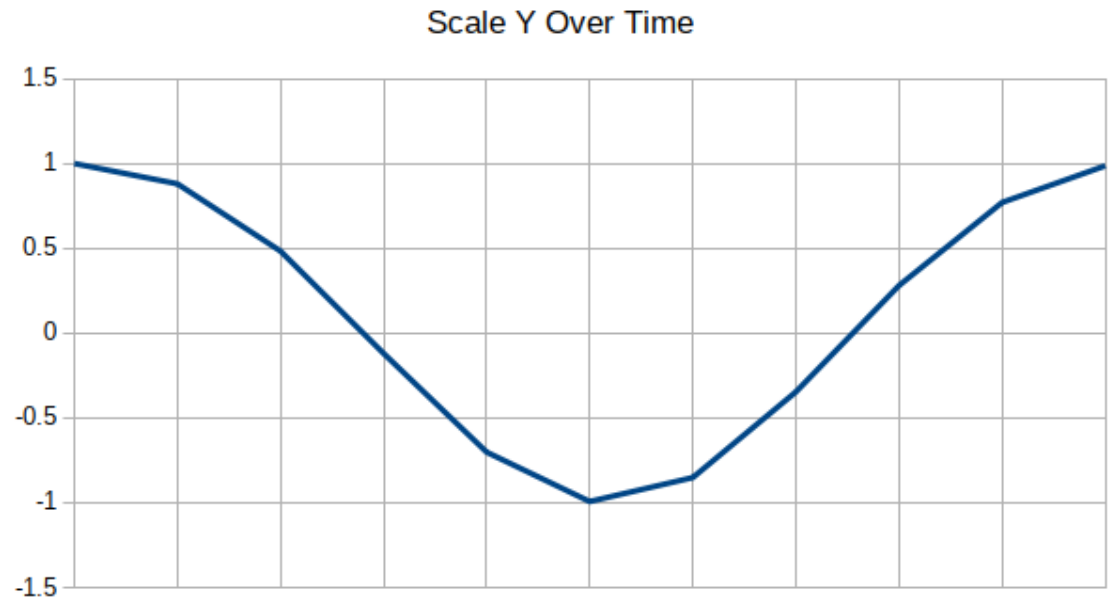


*Illustration 112: This illustration shows the change in scale in the Y plane over time in relation to the total size of the object.*

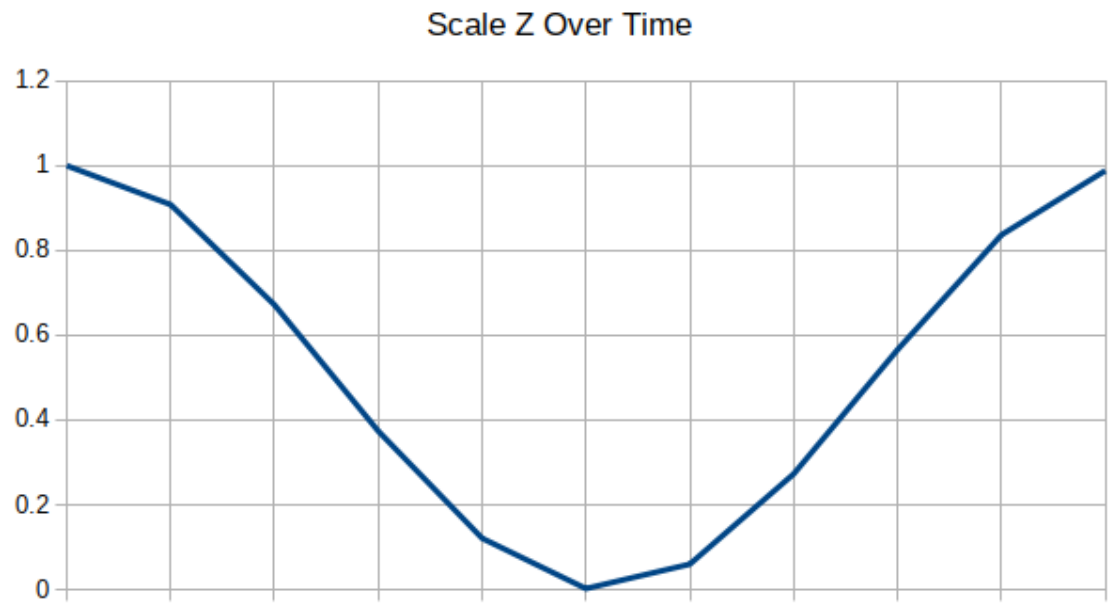


*Illustration 113: This illustration shows the change in scale in the Z plane over time in relation to the total size of the object.*

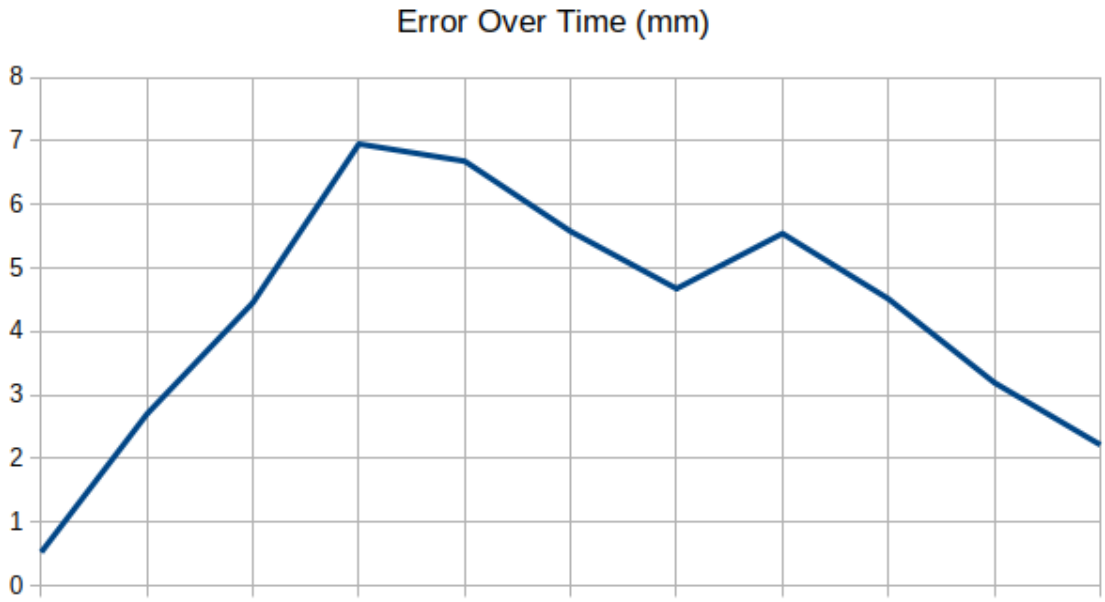


*Illustration 114: This illustration shows the change in error over time in millimetres.*

### Arduino Test: Overview

The final validation experiment tried was an experiment using data acquired using the Kinect application. The subject of the acquisition was a flat rotating object which was controlled using the experiment application, this flat rotating object moved at a maximum rate of one degree every second with a pause of one second for every one hundred and eighty degrees rotated.

The registration algorithm applied to this data was applied relative to the first point cloud meaning that each point cloud was registered to the first point cloud. For this experiment a region of the entire point cloud was tracked in an attempt to mimic the tracking of a surrogate or breathing signal, the region which was tracked was located at one of the corners of the flat rotating object. This means that the expected output from this application would be a general polynomial curve in any of the rotational, translation or scale variables which sines up and down at a periodic rate. This is because this experiment is only an initial proof of the functionality of the application and further experiments would be needed to verify exactly how and what is being tracked by the application, however if some form of surrogate or breathing signal could be extracted that would be a major positive for the application given the limited time that it was produced in.

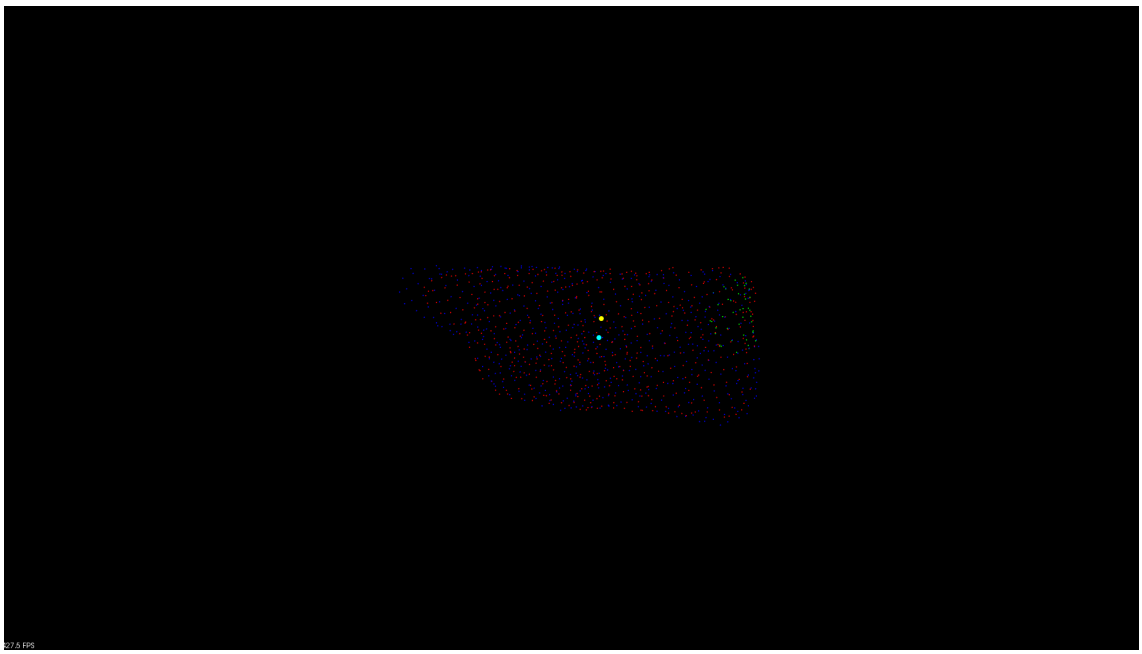

*Illustration 115: This illustration shows the output visualisation of the final pair of point clouds from the front.*

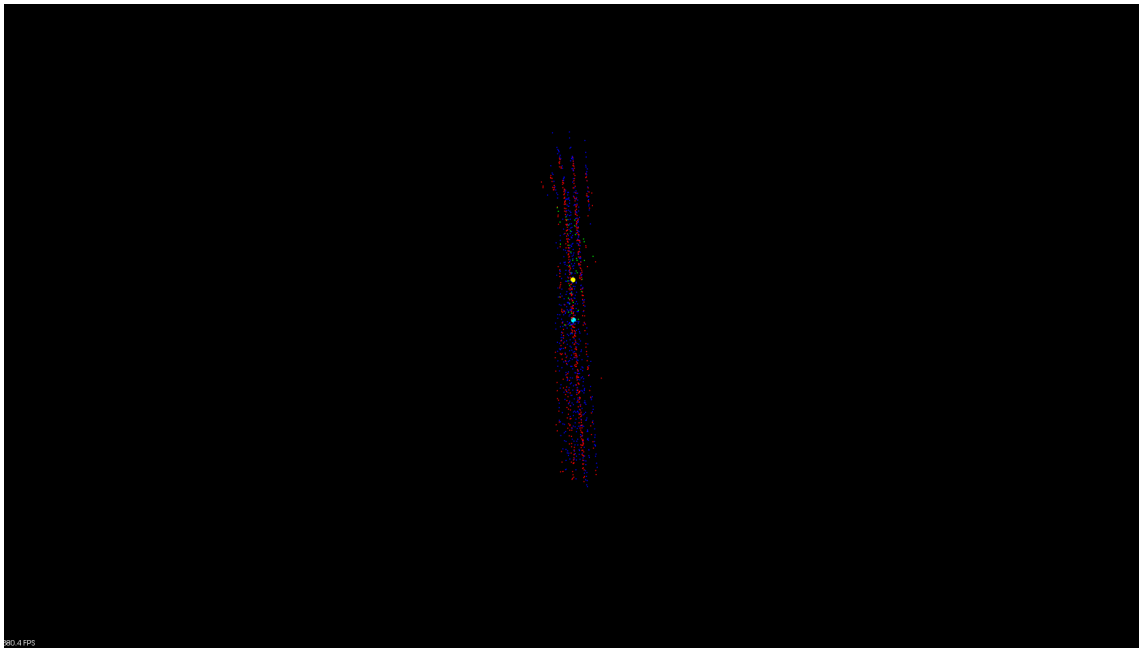

*Illustration 116: This illustration shows the output visualisation of the final pair of point clouds from the side.*

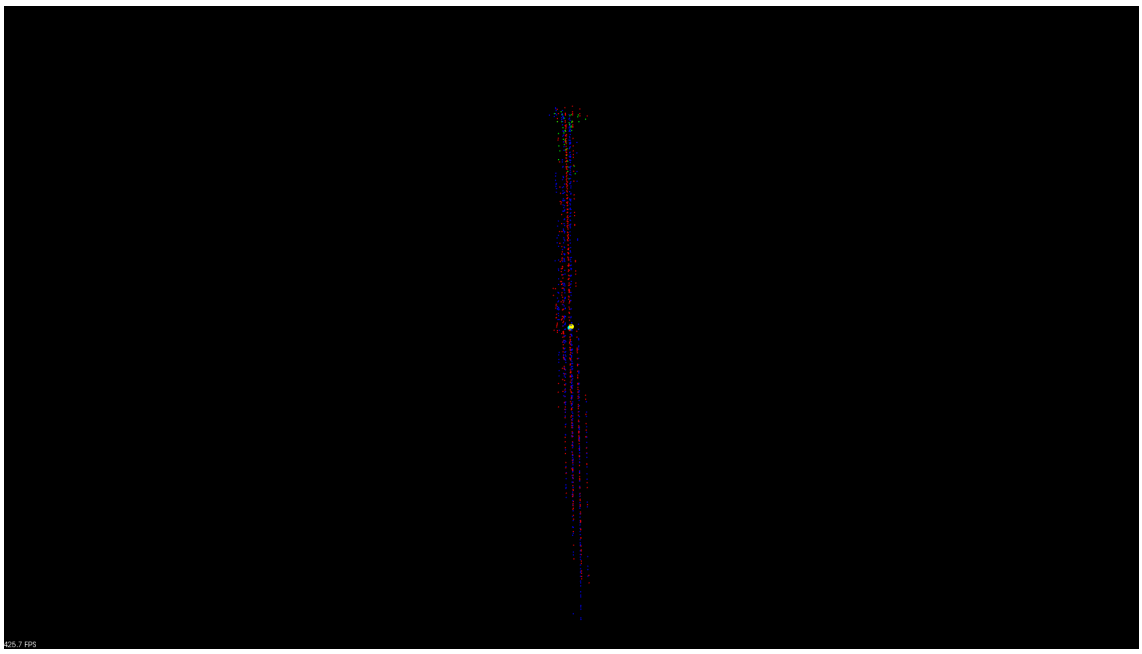

*Illustration 117: This illustration shows the output visualisation of the final pair of point clouds from the top.*

**Arduino Test: Results**

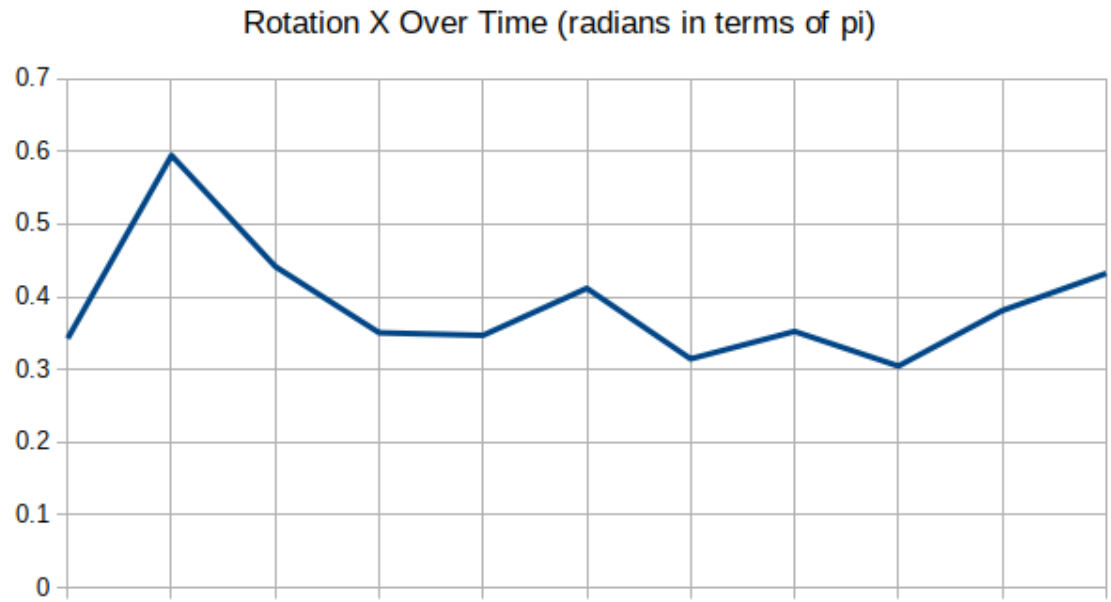


*Illustration 118: This illustration shows the change in rotation in the X plane over time in terms of Pi.*

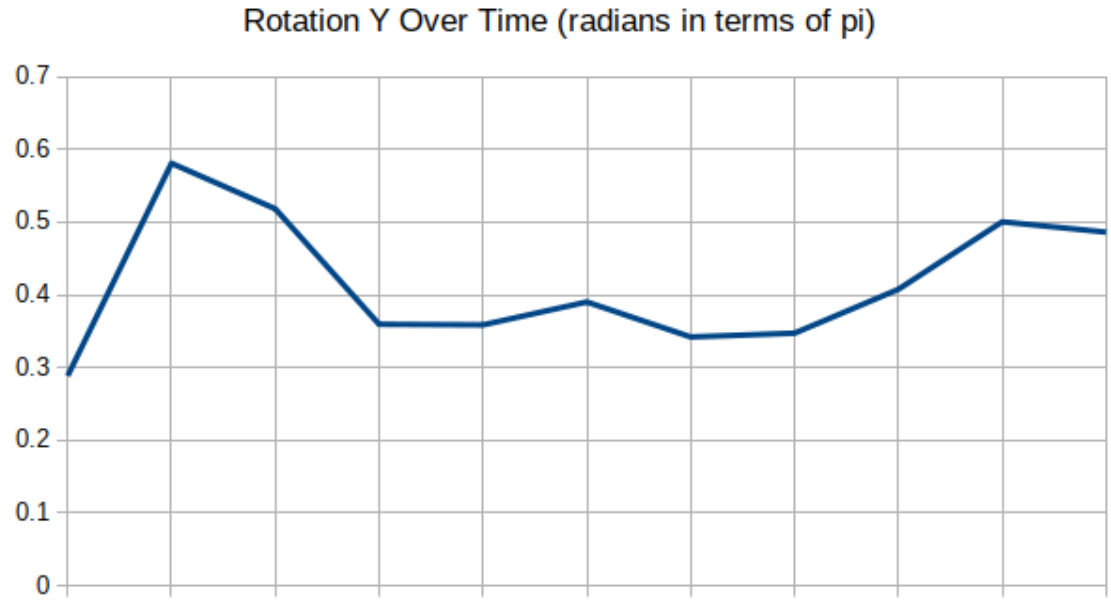


*Illustration 119: This illustration shows the change in rotation in the Y plane over time in terms of Pi.*

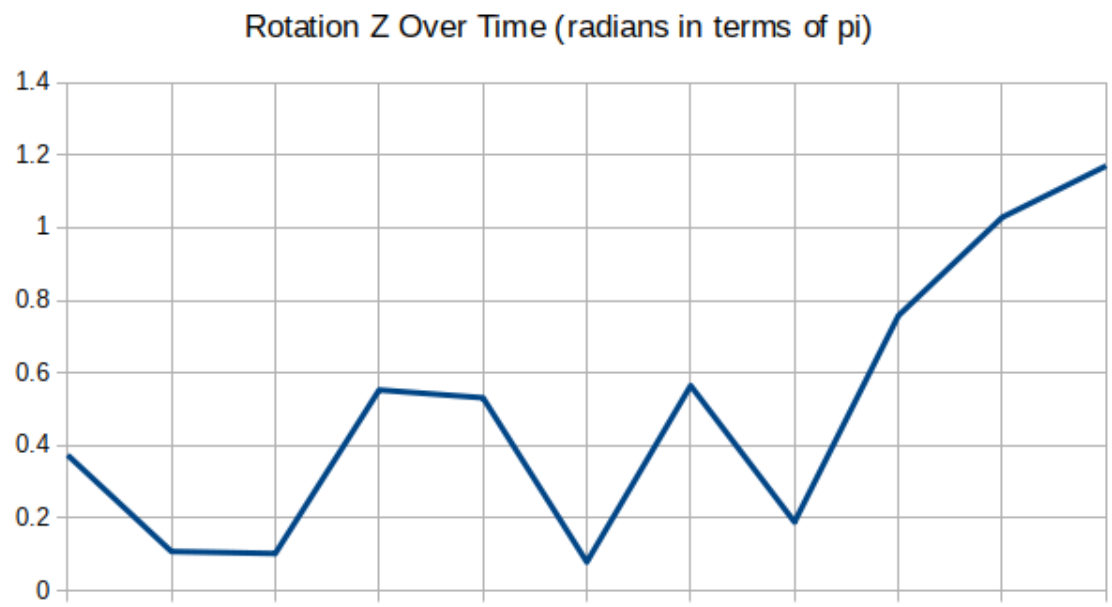


*Illustration 120: This illustration shows the change in rotation in the Z plane over time in terms of Pi.*

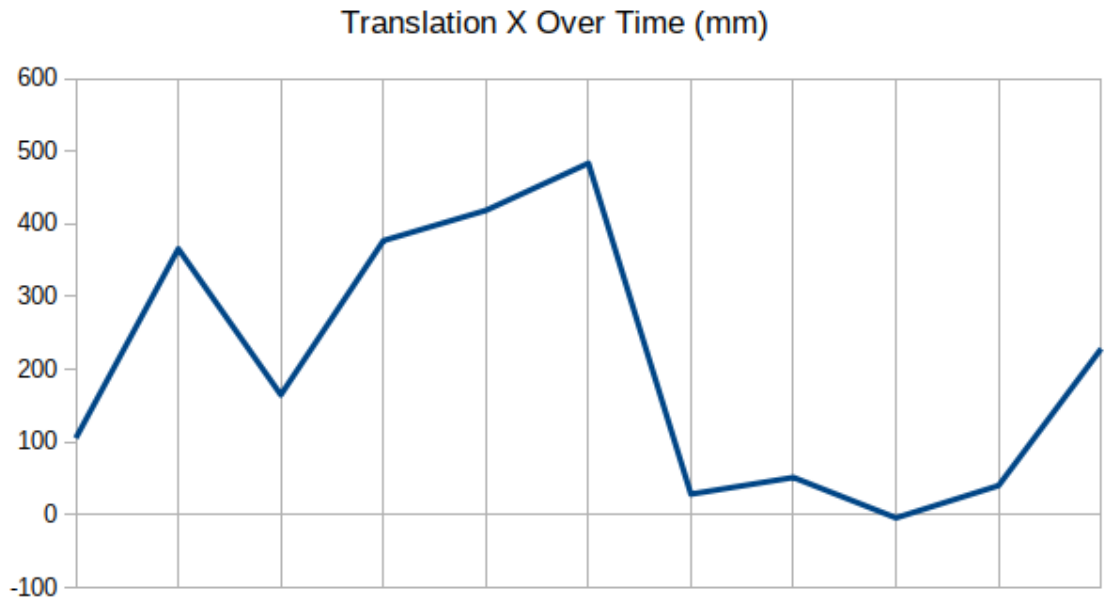


*Illustration 121: This illustration shows the change in translation in the X plane over time in millimetres.*

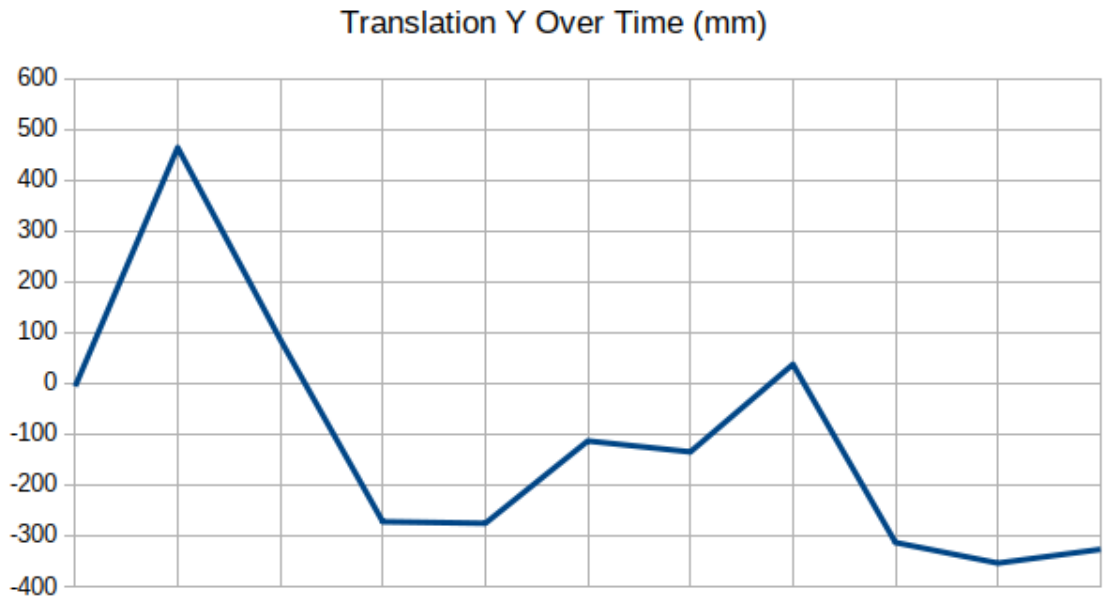


*Illustration 122: This illustration shows the change in translation in the Y plane over time in millimetres.*

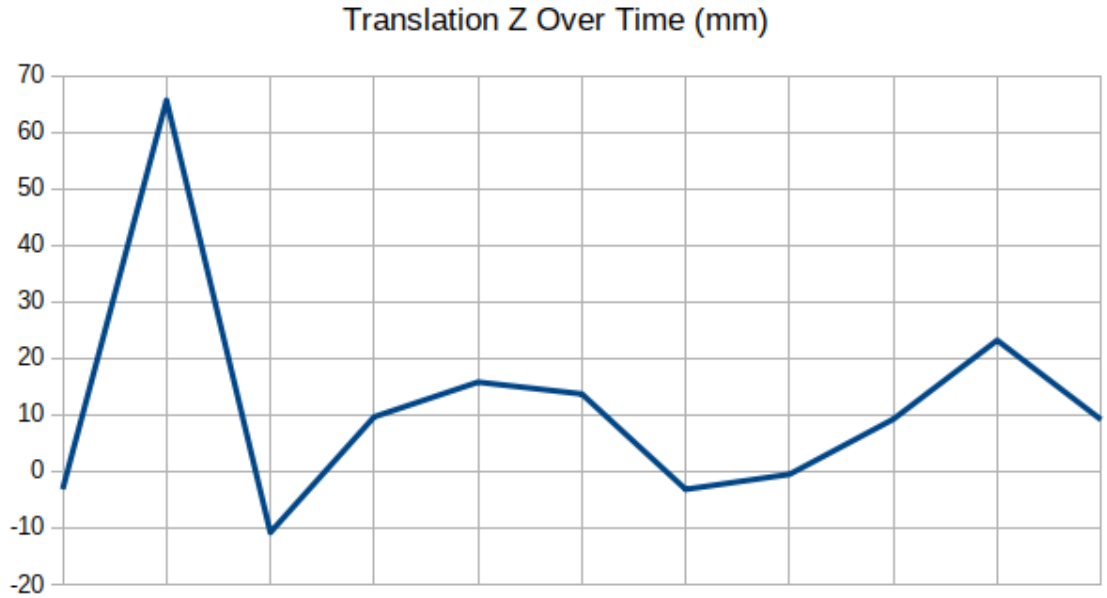


*Illustration 123: This illustration shows the change in translation in the Z plane over time in millimetres.*

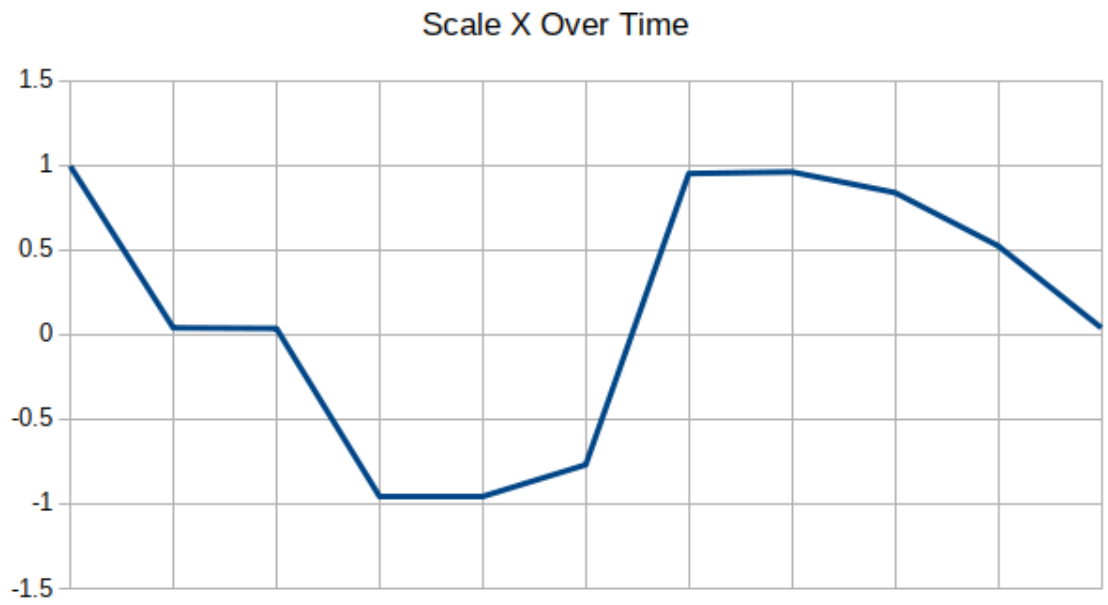


*Illustration 124: This illustration shows the change in scale in the X plane over time in relation to the total size of the object.*

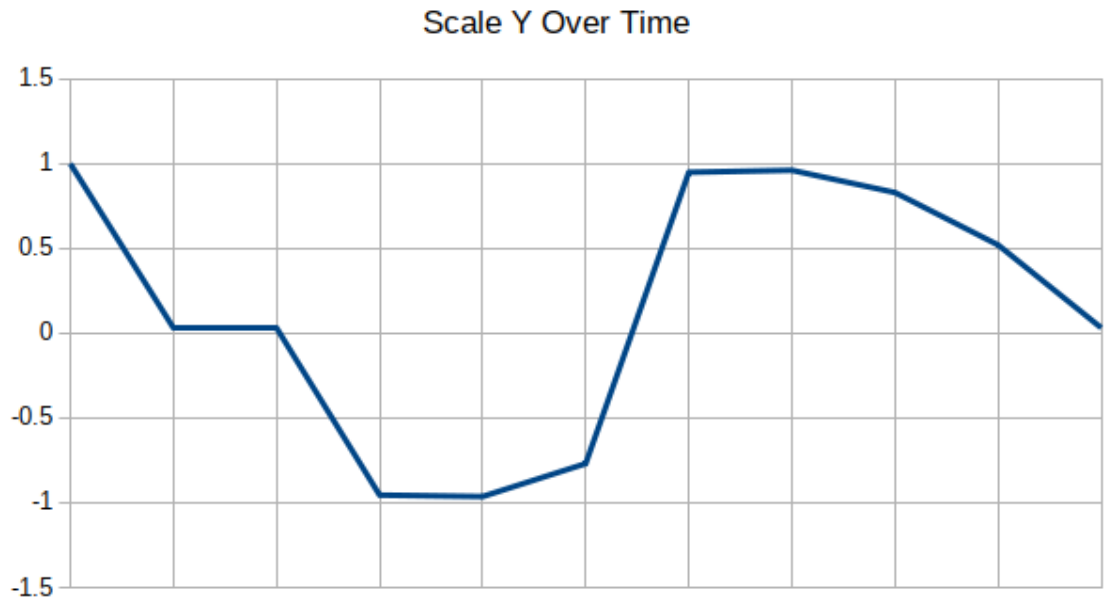


*Illustration 125: This illustration shows the change in scale in the Y plane over time in relation to the total size of the object.*

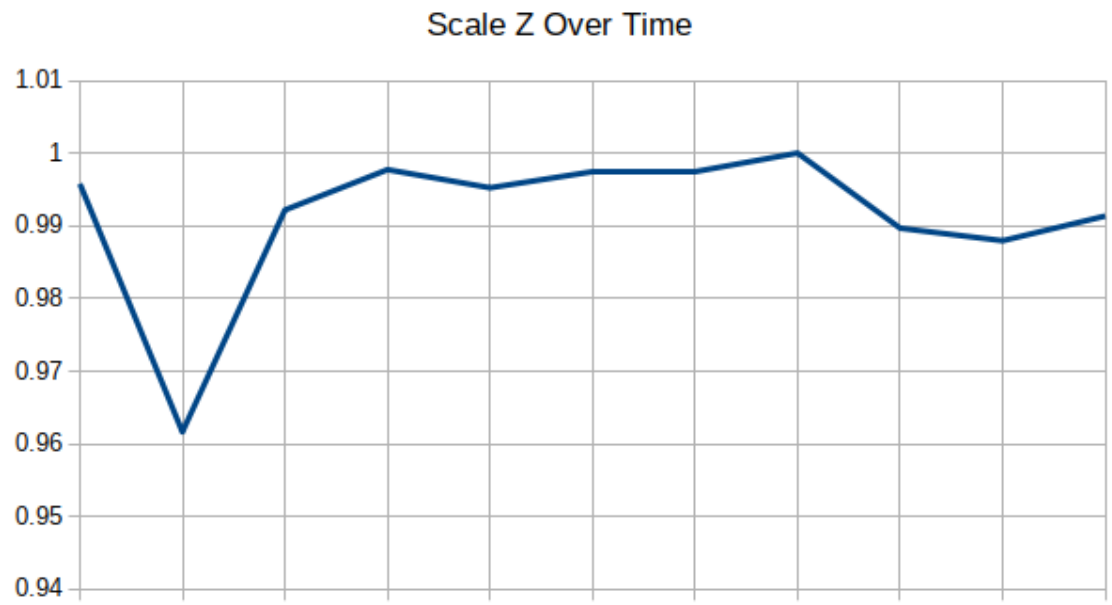


*Illustration 126: This illustration shows the change in scale in the Z plane over time in relation to the total size of the object.*

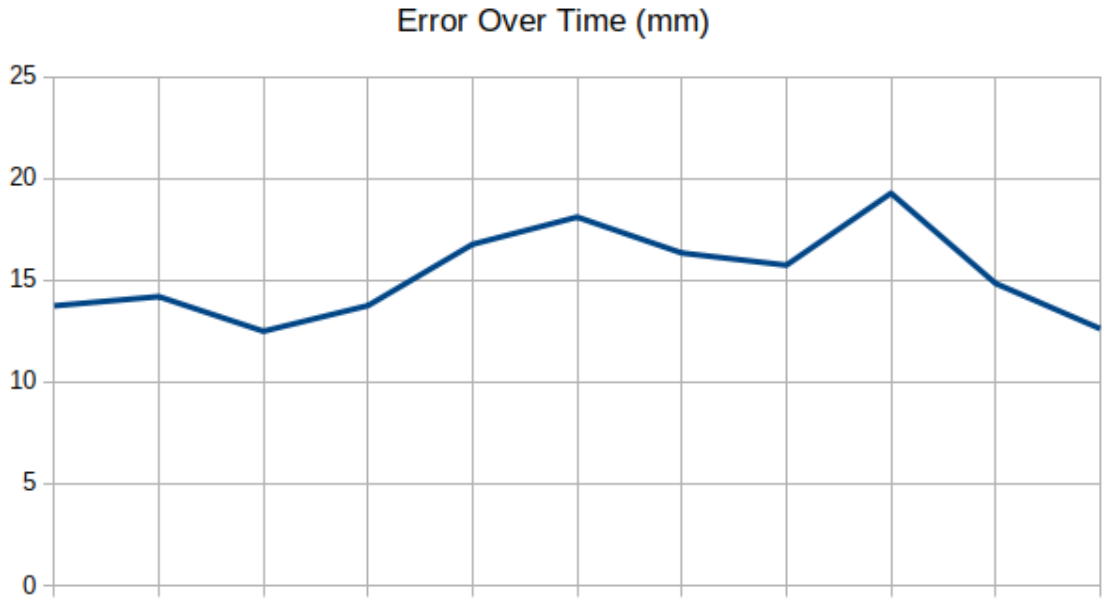


*Illustration 127: This illustration shows the change in error over time in millimetres.*

### Experiment Evaluation: Overview

As can be seen from the data above, generally the application can be regarded as being able to register two point clouds together, this is apparent particularly in the images of the visualisation of the outputs from the plane and curve experiments.

These images show the result of transposing the final point cloud from both experiments to the source point cloud from the experiment using the homogeneous transformation calculated during their respective registration steps. The final point cloud from both experiments had been transposed by at least a metre and rotated at least ninety degrees before being registered to their respective source point clouds, thus it is extremely positive that it appears in both images that the point clouds contained within them have been aligned sufficiently to be indistinguishable through the registration process. However, the same cannot be said initially for the Arduino experiment as the image of this experiment doesn't clearly show the tracked region.

Although it initially appears from the simple plane test graphs that the output rotation and translation vary wildly this only appears to be the case because the scale of the graphs are so small. The graphs for the relative registration show a continuous linear rotation and translation between each set of point clouds and the graphs for the continuous registration show a linear increase in the amount of rotation and translation from the first point cloud to the last point cloud.

The error from these graphs are generally below an acceptable threshold, however they do spike in places.

The graphs for the plane and curve test however are a lot more difficult to interpret. Because the objects are being rotated through a full one Pi radians from first to last point cloud it makes sense that the graphs for both experiments show an increase up to a full rotation before rotating back in the opposite direction, this is because the objects have rotational symmetry. What is more difficult to explain is the fact that the translation and scale seem to vary in unpredictable ways from first point cloud to last point cloud, the scale of the transformation appears to be affecting the translation of the point cloud in ways that are undesirable.

The scale of a subject in a PET/CT scanner is usually unchanged so to include a variable scale in the output from this application would not reflect the true movement of the subject in real life. Thus if the application was to be improved a registration algorithm that didn't scale the source and target point clouds would probably produce a more accurate result.

The error of these graphs are generally very high, as such further experiments should be conducted to re-enforce the claims made above. The error in these cases could be

the fact that the acquisition took place over such a long time and all registration took place back to the first point cloud.

The graphs for the Arduino test are initially difficult to interpret as the expected rotation and scale are unknown. However, what is positive is that a polynomial curve can clearly be seen in the output graphs meaning that the tracking algorithm must be tracking some of the rotational movement of the acquisition data, what is less positive is that the application appears to occasionally treat rotational movement as a combination of translations in the horizontal and vertical planes.

## 5.2 Project Achievements

In order for the project to be completed in the time frame discussed in the appendix below (Appendix F:) (Appendix J:) the aims which were laid out at the start of the report and in the appendix below (Appendix G:) included the objectives bellow.

The aim to read read data from the Kinect camera was a success, an OpenGL visualiser program was written to test this functionality. This aim took less time than expected as there was a lot of example code to build from.

The aim to create a stand alone application to interface with the Kinect camera was a success, an application which interfaced with the Kinect camera was written. This took longer than anticipated as the documentation which outlined the settings of the Kinect object from OpenKinect Libfreenect was more convoluted than expected.

The aim to add the ability to control the tilt motor of the Kinect camera was a success, a GUI element was added which allowed for the tilt motor to be incremented up and down by a fixed amount, a piece of code was also needed to stop the motor from defaulting to it's maximum or minimum value if multiple simultaneous commands were sent.

The aim to add the ability to extract depth and/or RGB images from the Kinect camera was a success, both depth and RGB images could be extracted by using the callbacks from the Kinect camera, however the RGB images turned out to be ultimately useless. This took longer than anticipated as the documentation which outlined the settings of the Kinect object from OpenKinect Libfreenect was more convoluted than expected.

The aim to add the ability to save depth and/or RGB images from the Kinect camera to text or binary file was a success, however the RGB images turned out to be ultimately useless.

The aim to add the ability to output multiple depth and/or RGB images per execution of the application was a success. This took less time than expected as hardly any extra code was needed to convert from outputting one file to outputting multiple files.

The aim to add the ability to timestamp output with a timestamp relative to the Kinect camera was a success, however a timestamp was also needed that was relative to epoch. This took less time than expected as the timestamp from the Kinect camera was readily available.

The aim to add the ability to change output settings for the depth and/or RGB images was a success. An element was added to the GUI which skipped the output of unneeded files.

The aim to add the ability to output header files for the depth and/or RGB images was a success. This took less time than expected as the code had already been written to output files.

The aim to create a stand alone application to load the header output from the Kinect camera application. The Kinect application was copied in order to cut down on the initial set up of the application. This took longer than anticipated as the parser to load the KPCLP headers was written to handle any data type, this was unnecessary but is a nice feature to have.

The aim to add the ability to load data from header file was a success. This took longer than anticipated as an unforeseen bug occurred with the way that the binary files interlaced the columns of the depth images was only noticed at this point in the development.

The aim to add the ability to calculate point clouds from the loaded data was a success. This aim was completed using a calculation which originated from the OpenKinect Libfreenect documentation. This took longer than anticipate as many different approaches were discussed before the OpenKinect Libfreenect solution was accepted.

The aim to add the ability to calculate centre of gravity for the point clouds was a success. This took less time than expected as the PCL contained a method to do this.

The aim to add the ability to visualise the point clouds and centres of gravity was a success, however the libraries which this code depends on contains bugs which affect this solution and cannot be circumnavigated. This took less time than expected as the PCL contained a method to do this.

The aim to add the ability to globally threshold the point clouds was a success. The code which calculated the point clouds from the depth images was edited to fill any point past a certain depth with NAN data.

The aim to add the ability to de noise the point clouds was a success. This took less time than expected as the PCL contained a method to do this.

The aim to add the ability to downsample the point clouds. This took less time than expected as the PCL contained a method to do this.

The aim to add the ability to register between the point clouds was a success, however the libraries which one of the registration algorithms depends on contains bugs which affect this solution and cannot be circumnavigated. This took less time than expected as the PCL contained a method to do this.

The aim to add the ability to use the output of previous registration step as an input to the next registration step was a success. This took longer than anticipate as the entire application needed to be refactored to allow for this functionality.

The aim to add the ability to extract motion signal from the registration output was a success. This this took less time than expected as the registration algorithm outputs a homogeneous transformation.

The aim to add the ability to asses the amount of movement between the point clouds was a success. This this took less time than expected as the registration algorithm outputs a homogeneous transformation.

The aim to add the ability to skip processing based on the amount of movement between the point clouds was a success, however only Euclidean linear distance movement and not eigenvector movement was implemented.

The aim to add the ability to extract motion signal from the movement between the point clouds was a success. This took less time than expected as the previous aim calculated this motion signal.

The aim to add the ability to track a region of the point clouds was a success, however the tracking methods from the PCL were not implemented. This took longer than anticipated as multiple different tracking algorithms were implemented.

The aim to add the ability to extract the motion signal from the tracked region of the point clouds was a success. This took less time than expected as the registration algorithms which had already been implemented were used to implement this functionality

The aim to add the ability to change the output settings and values for the registration step and the motion signal extraction was a success. This took longer than anticipated due to the sheer amount of code needed to add settings for every value in the program.

The aim to add the ability to save the values and output of the registration step and the motion signal extraction to a text or binary file was a success. This took longer than anticipated as the format of the output was changed multiple times

The old aim to port the SAvVy framework to Windows was a success, however it turned out to be entirely unrelated to this project.

The old aim to add the ability to synchronise the timestamps contained within the point clouds that were output from the Kinect camera to the output from the PET scanner was a failure. This area was researched fully, however because the manufacturers of the PET/CT scanners have their devices locked down so tightly no progress was able to be made on this aim.

The old aim to add the ability to translate the origin of the space used by the Kinect camera to match the space used by the PET scanner was a failure. This area was researched fully, however because the manufacturers of the PET/CT scanners have their devices locked down so tightly no progress was able to be made on this aim.

The old aim to add the ability to apply the motion calculated from the output of the Kinect camera to the output of the PET scanner was a failure. This was a failure because the Kinect camera was unable to be temporally or spatially synchronised to the PET/CT scanner.

The old aim to port the application to Windows was unnecessary as the application was entirely cross platform by design.

The old aim to add the application to the windows version of the SAvVy framework as a library was unsuccessful as time had ran out before this could be attempted.

If this project were to be attempted again or more time was to be spent working on the application produced then areas which should be improved include obviously firstly working on the areas highlighted above where work was incomplete or failed entirely.

Another area that could be improved would be to add the ability to threshold point clouds based not only upon removing points beyond a threshold but also points closer than another threshold, this would allow for the ROI to be defined more accurately.

Furthermore, in order to lower the error when attempting to extract a surrogate or breathing signal usually the registration style is set to register to the mean point cloud rather than the first point cloud. (McClelland *et al.*, 2017)

Although tracking algorithms have been implemented the tracking algorithms from the PCL library should be added.

Finally, there are buttons on the GUI of the application which currently do not perform their function, like the ability to save the contents of the log, these elements should be added.

# 6 Conclusion

The project in its current state has met most if not all of the objectives that were initially set out for it. The software has had most of the functionality that it depends on implemented and it has been implemented to a decent standard with a suitable user interface.

There are some known bugs in the application, however, these bugs do not interfere with the overall output of the application and with a small amount of additional work could be remedied.

Based upon the conclusions drawn from the experiments conducted in the evaluation section the null hypothesis of this project can be rejected and the hypothesis accepted.

# Appendix A: Research Proposal

A stand-alone cross-platform framework for image analysis, reconstruction and data processing of a variate of data acquired by PET/SPECT scanners and potentially MRI is under development. The framework links and makes use of external open source libraries in order to handle data formats and perform various complex tasks. The GUI is developed on QT and there is a strong drive to have cross-platform compatibility (especially between Windows and Linux).

The goal will be to provide a responsive/neat/intuitive plug-in able to acquire data from a Microsoft Kinect camera in an easy plug and play fashion and extract from those warp spaces and/or motion signals for motion corrected or compensated medical image reconstruction.

This is a project on scientific programming but many programming challenges have to be addressed. The successful candidate will need to fluent in a C variance, with drive to improve his skills on C++ and Cmake.

# Appendix B: Ethics Report

If your project uses other people ('participants') for the collection of information (typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system) then you need to read through the checklist in Section A below before completing the declaration in Section B.

If your project does **not** make use of other people then you can skip Section A and directly complete the declaration in Section B by marking box '1' with an X.

## Section A

### 1. Participants will not be exposed to any risks greater than those encountered in their normal working life.

Researchers have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

### 2. The experimental materials will be paper-based, or comprised software running on standard hardware.

Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and PDAs is considered non-standard.

### 3. All participants will explicitly state that they agree to take part, and that their data could be used in the project.

If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. Each participant should sign a separate consent form.

Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives will be offered to the participants.

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

5. No information about the evaluation or materials will intentionally be withheld from the participants.

Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant will be under the age of 16.

Parental consent is required for participants under the age of 16.

7. No participant will have an impairment that may limit his or her understanding or communication.

Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants will be informed that they can withdraw at any time.

All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants will be informed of my contact details.

All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module coordinator or supervisor as part of the debriefing.

## 11. The evaluation will be discussed with all the participants at the end of the session, and all participants will have the opportunity to ask questions.

The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.

## 12. All the data collected from the participants will be stored in an anonymous form.

All participant data (hard copy and soft-copy) should be stored securely, and in anonymous form.

If your evaluation does comply with all the twelve points above, please mark box ‘2’ in Section B.

If your evaluation does not comply with one or more of the twelve points above, please mark box ‘3’ in Section B unless you **know** that your supervisor already has ethical approval for the project (in which case mark box ‘4’). If you are unsure mark box ‘3’.

[Adapted from Department of Computing Science University of Glasgow Ethics checklist form for 3rd/4th/5th year, MSc IT/CS/ACS projects 2007]

## Section B

| | |
|---|---|
| Student Name | Alexander C Whitehead |
| Project Title | Motion Signal Extraction Framework for the Microsoft Kinect Camera: Point Cloud Registration and its Application as a Motion Correction Metric in PET/CT |
| Supervisors Names | Dr D Parker<br>Dr N Efthymiou |

| This is a declaration that the ethical concerns for above project have been considered (in particular with regards to the 12 point checklist above) with the following outcome: | | Please mark only ONE box with an X |
|---|---|---|
| 1 | This project does not involve other people in the collection of information and therefore does not require an ethical review | X |
| 2 | This project complies with the **entire** twelve point ethical checklist and therefore does not require ethical review. | |
| 3 | This project does not comply with **all** of the twelve points above and therefore does require ethical review and the completion and submission of an ethical approval form. | |
| 4 | This project does not comply with **all** of the twelve points above, however the supervisor already has ethical approval for this research | |

If you have marked box '3' you will be expected to apply for ethical approval. Further advice is available from both your project supervisor and the Department's Ethical Officer, as well as by reading and completing this form.

## Appendix C: Risk Assessment

| **Risk** | **Current Risk** | | | **How to Avoid** | **How to Recover** | **Residual Risk** | | |
|---|---|---|---|---|---|---|---|---|
| | Severity (L/M/H) | Likelihood (L/M/H) | Significance (Severity, Likelihood) | | | Severity (L/M/H) | Likelihood (L/M/H) | Significance (Severity, Likelihood) |
| Data loss | H | M | HM | Keep backups | Reinstate from backups | L | M | LM |
| Loss of backups | H | L | HL | Keep multiple backups on multiple different forms of media in multiple different locations | Use alternate backup | L | L | LL |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Underrate workload | H | M | HM | Regularly review progress against Time Plan | Invest more time into work or reduce objectives | H | L | HL |
| Critical error in deliverable | H | M | HM | Perform adequate background research | Thoroughly test and debug code | H | L | HL |
| Skill risk | M | M | MM | Perform adequate training or seek out specialists | Invest more time into background research | L | L | LL |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Development materials unavailable | H | L | HL | Develop a contingency plan in order to be able to pivot the application to another domain | Pivot the application to another domain | L | L | LL |
| Scope creep | M | H | MH | Fully define objectives | Fully define current objectives and do not change the objectives again | M | L | ML |
| Inefficient program speed | H | L | HL | Invest time in testing and debugging code | Optimise code or remove slow code | M | L | ML |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Medical emergency | H | L | HL | Care for developers health including regular periods of rest | Comment code regularly so that it is well understood | M | L | ML |
| Injury while using application | H | M | HM | Ensure proper training is in place before the application is used | Medical attention should be available to anyone injured while the application is in use | H | L | HL |
| Damage to equipment while the application is in use | H | M | HM | Test application compatibility with different working situations | Repair damage to equipment | H | L | HL |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Incorrect conclusions drawn because of use of application | H | H | HH | Never solely use the application to draw conclusions | Use other sources of data to aid in drawing conclusion | M | M | MM |

*Table 1: This table shows a breakdown of some risks which could occur while working on the project. These risks are then expanded upon by determining the current risk, determining solutions to these risks and then determining the residual risk.*

# Appendix D: Final Task List

| # | Task Name | Description | Duration (days) |
|---|---|---|---|
| 1 | Research PET Scanners | Conduct background research specifically related to the operation of PET scanners.<br>This research will aid in the writing of the initial report and the design of the application. | 29 |
| 2 | Research Alternative Solutions | Conduct background research specifically related to alternative solutions to motion correction of PET data.<br>This research will aid in the writing of the initial report and the design of the application. | 22 |
| 3 | Research Kinect Camera | Conduct background research specifically related to the operation of the Kinect Camera and its API.<br>This research will aid in the writing of the initial report and the design of the application. | 22 |
| 4 | Research Point Cloud Library | Conduct background research related to the open source PCL, specifically its ability to perform the cleaning of point clouds and the registration of point clouds.<br>This research will aid in the writing of the initial report and the design of the application. | 22 |
| 5 | Write initial report | Write the initial report or design specification, this will involve writing the bulk of the content. | 26 |

| | | | |
|---|---|---|---|
| 6 | Edit initial Report | Edit the initial report or design specification, this will involve editing the content and ensuring it is presented to the best standard possible. | 12 |
| 7 | Read data from the Kinect | An application should be created which can read data from the Kinect camera in real time and visualise this data to the user.<br><br>The open source Kinect driver should be used for this application, this is because it allows for the development on and distribution to multiple different operating systems, opening up the eventual developer and user base for the application. | 8 |
| 8 | Create stand-alone application to interface with Kinect | A stand-alone console application and library should be created which can either be launched directly as an executable program or can be compiled as a library and then accessed within another application.<br><br>To achieve this the application could be written as a library and then a wrapper application could be written to run it independently.<br><br>This is advantageous as it allows for more rapid prototyping of ideas without having to rely on a large codebase, which in itself could contain errors, while also not hindering the ease with which the application could be integrated into a larger framework.<br><br>This application should be able to access the callbacks from the Kinect camera. | 15 |

| | | | |
|---|---|---|---|
| 9 | Add ability to control the tilt motor of the Kinect | The ability to send signals to the Kinect camera in order to request that it adjusts its tilt angle should be added, the ability to also request the current tilt angle should also be considered.<br><br>An element should be added to the GUI to allow for the user to request that the Kinect camera's tilt is adjusted up or down by a preset amount, the ability to input an exact angle for the Kinect to tilt to should also be considered.<br><br>The Kinect camera's tilt angle should never be allowed to exceed safe parameters.<br><br>This feature would be useful as it allows for the user to better define the area that the Kinect camera will be scanning. | 15 |
| 10 | Add ability to extract depth and/or RGB images from Kinect | The ability for the application to read depth and/or RGB data which has been written into a buffer by the callbacks from the Kinect camera should be added.<br><br>Initially the callbacks from the Kinect camera can operate on the main thread, however it would be ideal to access the callbacks from their own thread as this would not only speed up the application and allow for fewer dropped frames from the Kinect camera but would also stop the callbacks from blocking input to the GUI.<br><br>The depth and/or RGB data should be held in memory in order to allow for further processing. | 15 |

| | | | |
|---|---|---|---|
| 11 | Add ability to save depth and/or RGB images from Kinect to text or binary file | The ability to save the data from one scan of the Kinect camera should be added to the application.<br>The data should be saved to a file in storage as either a text file or a binary file.<br>An object should be created to represent the data output from the Kinect camera, this object should overload the streaming operators which will allow it to be written either automatically to the GUI or to a file. | 15 |
| 12 | Add ability to output multiple depth and/or RGB images per execution | The ability to save the data of multiple concurrent scans from the Kinect camera should be added to the application.<br>It is possible to either save each piece of data as they are read from the Kinect camera or to save all pieces of data at once after the scan has been complete.<br>Each new piece of data should be saved into a list or vector contained within the object created to represent the data output from the Kinect camera, this will allow the data to be accessed again at a later point in the program.<br>After execution has ended, a method could be called which would stream the entire list or vector either to the console output or to a file. | 8 |

| | | | |
|---|---|---|---|
| 13 | Add ability to timestamp output with timestamp relative to Kinect | The ability to save the time that a scan occurred should be added to the application.<br><br>Each timestamp should be saved within the object created to represent the data output from the Kinect camera, this will allow all data relevant to a given scan to be contained together.<br><br>Timestamping the data from the Kinect camera is necessary to synchronize the Kinect camera output to the PET scanner output later.<br><br>The timestamp saved should not only be relative to the Kinect camera but also relative to the time since epoch in order to be able to better identify data from different scans.. | 8 |
| 14 | Add ability to change output settings for the depth and/or RGB images | The ability to change the output directories for the depth and/or RGB images should be added.<br><br>The ability to block the operation of either depth and/or RGB data callbacks should be offered to allow for the user to optimise the operation of the application for their own needs. | 8 |
| 15 | Add ability to output header files for the depth and/or RGB images | The ability to generate header files for the depth and/or RGB images should be added, these header files can be used later when reconstructing point clouds from the depth data.<br><br>Each header file should contain at least the file path to the image file, information regarding the resolution of the image, the bit depth of the image, the time at which the image was acquired and how many dimensions there are to the data. | 15 |

| | | | |
|---|---|---|---|
| 16 | Create stand-alone application to load the header output from the Kinect application | A stand-alone console application and library should be created which can either be launched directly as an executable program or can be compiled as a library and then accessed within another application.<br><br>To achieve this the application could be written as a library and then a wrapper application could be written to run it independently.<br><br>This is advantageous as it allows for more rapid prototyping of ideas without having to rely on a large codebase, which in itself could contain errors, while also not hindering the ease with which the application could be integrated into a larger framework.<br><br>This application should be able to open the header files created by the Kinect camera application and load the information contained into an object representing the header file. | 15 |
| 17 | Add ability to load data from header file | The ability to load the depth and/or RGB data located at the file path specified in the header file should be added, this depth and/or RGB data should be loaded into a buffer which is located in the object that was created to represent the header file.<br><br>This data should be loaded at a point after the loading of the header file itself, this is because it can take a substantial amount of time to load the data from storage and as such the user should be certain that the correct header files have been selected before the data is extracted.<br><br>The size of the buffer to contain the data can be determined from the resolution and dimension information in the header file, the order that the data buffers should be loaded and stored can be determined from the timestamp located within the header file. | 15 |

| | | | |
|---|---|---|---|
| 18 | Research calculating point clouds from the loaded data | Conduct background research specifically related to the mathematics of how to convert from a 2D depth image to a 3D Cartesian point cloud.<br><br>This research will aid in the writing of the final report and the implementation of the application. | 22 |
| 19 | Add ability to calculate point clouds from the loaded data | The ability to calculate an unordered Cartesian point cloud from a depth image should be added.<br><br>Because perspective forces points to be displaced by greater amounts in the x and y axis as they are displaced in the z axis the algorithm to calculate a point cloud should take perspective into account when determining a points position, this calculation should use the focal distance of the camera in order to estimate the displacement caused by the camera. This approach is far more accurate than just displacing each pixel by its depth in the z axis.<br><br>If the Kinect camera cannot detect a valid depth for a given pixel in the depth image this should be represented by a NAN value in the point cloud, this is because it will be possible then to later iterate through the point cloud and remove any NAN values. This will not only reduce the amount of memory that is used by the application but will also speed up any subsequent calculations. | 22 |

| | | | |
|---|---|---|---|
| 20 | Add ability to calculate centre of gravity for the point clouds | The ability to calculate the centre of gravity of a point cloud as a centroid should be added.<br><br>This centre of gravity can be used later to aid in more naive calculations such as where the movement of the point cloud as a whole needs to be taken into account.<br><br>The centre of gravity of the point cloud can also be used as part of a visualisation to aid in the users understanding of the data that they have acquired, the centre of gravity could also be used to aid in the users assessment of how changes to the settings and parameters of the program affect the programs output. | 8 |

| | | | |
|---|---|---|---|
| 21 | Add ability to visualise the point clouds and centres of gravity | The ability to visualise the point cloud pairs post registration and the centre of gravity of these point clouds should be added.<br><br>This visualisation can be used to aid in the users understanding of the data that they have acquired, the visualisation could also be used to aid in the users assessment of how changes to the settings and parameters of the program affect the programs output.<br><br>The visualisation should be displayed for each registered pair of point clouds and should block the programs execution while running. The visualisation should be optional so as to allow the application to be run without user input.<br><br>This method should be extendable so that any number of point clouds and centres of gravity can be added at a later date.<br><br>The colours of the point clouds and centres of gravity should be able to be set externally so as to ensure that each point cloud and centre of gravity is distinguishable from each other, the point size of the point clouds and centres of gravity could also be set externally so as to tailor the visualisation to each users needs. | 15 |
| 22 | Research point cloud processing | Conduct background research specifically related to the mathematics of noise reduction and downsampling of point clouds.<br><br>This research will aid in the writing of the final report and the implementation of the application. | 8 |

| | | | |
|---|---|---|---|
| 23 | Add ability to globally threshold point clouds | The ability to remove structures that are not ROIs should be added to the application.<br><br>In every set of data from the Kinect camera it is likely there will be structures that are of no importance to the functionality of the application, for instance the bezel of the PET scanner. These undesirable objects should be removed to cut down on processing power and memory usage. A smaller data set is also easier to visualise clearly.<br><br>There are multiple ways to achieve this result, a more naive method would be to define the ROI as being at a certain distance from the camera, any points that are beyond this threshold should be removed. A more complex solution would be to acquire a point cloud before scanning commences that only contains the objects that are to be removed from the data, therefore by subtracting the initial point cloud from every subsequent point cloud the bezel of the PET scanner and other extraneous undesirable data or ROIs will be removed.<br><br>The parameters of the threshold or segmentation algorithm should be set externally so as to allow the user to optimise the threshold or segmentation algorithm to their own needs. | 8 |

| | | | |
|---|---|---|---|
| 24 | Add ability to de-noise point clouds | The ability to remove extraneous or outlying points which represent added noise from the data should be added to the application.<br><br>This could be achieved by iterating through every point in a given point cloud and finding for a selection of points around said initial point the standard deviation of this group of points, if the initial point is outside a given standard deviation of this group of points it should be considered to be noise and should be removed from the point cloud as a whole.<br><br>The PCL contains methods and algorithms that have been implemented and tested to clean or de-noise point clouds.<br><br>The parameters of the de-noising algorithm, including the size of the group of points and the number of standard deviations that a given point can be from the group of points, should be set externally so as to allow the user to optimise the intensity and speed of the de-noising algorithm to their own needs. | 8 |
| 25 | Add ability to downsample point clouds | The ability to downsample point clouds should be added to the application.<br><br>The raw point cloud is likely to contain many more points than are necessary for the registration algorithm to accurately register point cloud to point cloud, for each unnecessary point in each given point cloud the time of execution of the application increases dramatically. Therefore, it is advantageous to downsample each point cloud, meaning to pass a filter, of a given voxel size, over the point cloud and to average every point inside the voxel to one individual point.<br><br>The size of the voxel used to downsample each point cloud should be set externally so as to allow the user to optimise the severity of the downsampling algorithm to their own needs. | 8 |

| | | | |
|---|---|---|---|
| 26 | Research point cloud registration | Conduct background research specifically related to the mathematics of how to register one point cloud to another and then how to extract the transformation needed to register one point cloud to another.<br><br>This research will aid in the writing of the final report and the implementation of the application. | 43 |
| 27 | Add ability to register between point clouds | The ability to translate one set of points to another should be added to the application.<br><br>The registration process can be either rigid or non-rigid; this means that either the relationship between the points must remain the same for rigid registration or the relationship between the points can change for non-rigid registration. Generally, a rigid registration will be applied to a pair of point clouds before a non-rigid transformation is applied in order to increase the accuracy of the non-rigid registration.<br><br>The PCL contains methods and algorithms that have been implemented and tested to register point clouds.<br><br>The parameters of the registration algorithm should be able to be set externally so as to allow the user to optimise the registration algorithm to their own needs, also the exact registration algorithm used should be selectable. | 15 |

| | | | |
|---|---|---|---|
| 28 | Add ability to use output of previous registration as input to next registration | The ability to use the output of the previous registration step as the guess matrix for the next registration step should be added to the application.<br><br>This means that each subsequent registration step should start from the transformation matrix that was determined in the previous registration step. This is advantageous because ideally an object is being tracked and as such will only move by a small amount from frame to frame, thus by starting each registration step from the end of the previous registration step fewer iterations of the registration algorithm should be required and subsequently the entire registration algorithm should compute faster and more accurately.<br><br>The parameters of the guess matrix should be able to be set externally so as to allow the user to optimise the registration algorithm to their own needs, also the ability to use the output of one registration step as the input to the next should be selectable. | 8 |
| 29 | Add ability to extract motion signal from registration output | The ability to extract the transformation matrix or vector field from the registration algorithm should be added to the application.<br><br>The transformation matrix which is extracted from the rigid registration algorithm represents the main variable to be used in the motion correction of the PET data, this matrix represents any scale, skew, rotation or translation that has occurred between the source and target point cloud as a homogeneous matrix.<br><br>If a non-rigid registration algorithm is used a vector field with a transformation matrix for each individual point should be output instead.<br><br>This transformation matrix or vector field should be stored in some kind of buffer in order to be used in other parts of the application. | 8 |

| | | | |
|---|---|---|---|
| 30 | Add ability to asses amount of movement between point clouds | The ability to extract the distance between the centre of gravity of both point clouds or the ability to extract the amount of movement from both point clouds should be added to the application.<br><br>There are two main methods which could be used to calculate the amount of movement between two point clouds, a naive approach would be to determine the linear distance between the centre of gravity of both point clouds, this could be calculated using the distance formula derived from the Pythagorean theorem. A more complex approach would be to use Eigenvectors.<br><br>The algorithm used to estimate the amount of movement between the two point clouds should be selected externally. | 8 |
| 31 | Add ability to skip processing based on amount of movement between point clouds | The ability to skip the registration step of the program based on a threshold on the amount of linear movement from point cloud to point cloud should be added to the application.<br><br>This is advantageous as a great deal of the overall processing time of the application will be dedicated to the registration step. Therefore, where the linear movement between a point cloud pair is determined to fall below an acceptable threshold the registration step could be skipped in order to speed up the application. This would be especially useful where the data set which is to be registered is exceptionally large.<br><br>The parameters of the movement threshold should be able to be set externally so as to allow the user to optimise the amount of movement which will skip the registration steps execution to their own needs. | 15 |

| | | | |
|---|---|---|---|
| 32 | Add ability to extract motion signal from movement between point clouds | The ability to extract a motion signal from the difference between the centre of gravities of the two point clouds should be added.<br><br>The difference between the positions of the centre of gravities of two point clouds, represented as a vector, may very naively estimate a kind of surrogate breathing signal for the entire subject.<br><br>The vector containing the difference between the centre of gravities of the two point clouds should be added to a buffer in order to be used in other parts of the application. | 15 |
| 33 | Research point cloud tracking | Conduct background research specifically related to the mathematical application of how to track a region of a point cloud over time and through deformation.<br><br>This research will aid in the writing of the final report and the implementation of the application. | 43 |
| 34 | Add ability to track a region of the point clouds | The ability to track a specific region of a point cloud should be added to the application.<br><br>A naive way to track a region of a point cloud would be, firstly to define a point at some distance away from the centre of gravity of the point cloud, then threshold the point cloud where every point in the point cloud where the linear distance to the point which was defined in the previous step is below the threshold is added to the ROI.<br><br>A more complex method to track a region of a point cloud would be to define some surface in the first point cloud, then using feature matching attempt to locate this surface in every subsequent point cloud.<br><br>The parameters of the tracking algorithm should be able to be set externally so as to allow the user to optimise the exact region which should be tracked to their own needs. | 29 |

| | | | |
|---|---|---|---|
| 35 | Add ability to extract motion signal from the tracked region of the point clouds | The ability to extract a motion signal from the difference between the tracked regions of the point clouds should be added.<br><br>The difference between the positions of the centre of gravities of the tracked regions of the point clouds as a vector may estimate the surrogate breathing signal of a region of the subject.<br><br>The vector containing the difference between the tracked regions of the point clouds should be added to a buffer in order to be used in other parts of the application. | 8 |
| 36 | Add ability to change output settings and values for registration and motion signal extraction | The ability to change the parameters of the application should be added to the GUI of the application.<br><br>The parameters of the application should be able to be set externally in the GUI of the application so as to allow the user to optimise the application to their own needs. | 8 |

| | | | |
|---|---|---|---|
| 37 | Add ability to save values and output of registration and motion signal extraction to a text or binary file | The ability to save the output from the entire registration application should be added to the application.<br>The data should be saved either to a file in storage as a text file or a binary file.<br>An object should be created that can contain the output string from the registration application, this object should overload the streaming operators allowing it to be written either automatically to the GUI or to a file.<br>The ability to change the output directories for the output should be added.<br>The ability to block the output of either text or binary output should be offered to allow for the user to optimise the operation of the application for their own needs.<br>A header should be added to the output from the registration application, this header should contain the parameters which were used in that particular set of calculations, this should allow experiments to be repeated and conclusions to be more accurately drawn. | 22 |
| 38 | Write final report | Write the final report or evaluation report, this will involve writing the bulk of the content. | 32 |
| 39 | Edit final report | Edit the final report or evaluation report, this will involve editing the content and ensuring it is presented to the best standard it could be. | 8 |

*Table 2: This table shows the final list of tasks that must be completed to finish the project. These tasks are then expanded upon by providing a short description of each task and an estimated duration in weeks.*

# Appendix E: Final Time Plan Table

| | | | |
|---|---|---|---|
| Research PET Scanners | 31/05/18 | 29 | 28/06/18 |
| Research Alternative Solutions | 07/06/18 | 22 | 28/06/18 |
| Research Kinect Camera | 07/06/18 | 22 | 28/06/18 |
| Research Point Cloud Library | 07/06/18 | 22 | 28/06/18 |
| Write initial report | 31/05/18 | 26 | 25/06/18 |
| Edit initial Report | 14/06/18 | 12 | 25/06/18 |
| Read data from the Kinect | 28/06/18 | 8 | 05/07/18 |
| Create stand-alone application to interface with Kinect | 28/06/18 | 15 | 12/07/18 |
| Add ability to control the tilt motor of the Kinect | 05/07/18 | 15 | 19/07/18 |
| Add ability to extract depth and/or RGB images from kinect | 12/07/18 | 15 | 26/07/18 |
| Add ability to save depth and/or RGB images from Kinect to text or binary file | 12/07/18 | 15 | 26/07/18 |
| Add ability to output multiple depth and/or RGB images per execution | 19/07/18 | 8 | 26/07/18 |
| Add ability to timestamp output with timestamp relative to Kinect | 19/07/18 | 8 | 26/07/18 |
| Add ability to change output settings for the depth and/or RGB images | 19/07/18 | 8 | 26/07/18 |
| Add ability to output header files for the depth and/or RGB images | 26/07/18 | 15 | 09/08/18 |
| Create stand-alone application to load the header output from the Kinect application | 26/07/18 | 15 | 09/08/18 |
| Add ability to load data from header file | 26/07/18 | 15 | 09/08/18 |
| Research calculating point clouds from the loaded data | 05/07/18 | 22 | 26/07/18 |
| Add ability to calculate point clouds from the loaded data | 05/07/18 | 22 | 26/07/18 |
| Add ability to calculate centre of gravity for the point clouds | 02/08/18 | 8 | 09/08/18 |
| Add ability to visualise the point clouds and centres of gravity | 02/08/18 | 15 | 16/08/18 |
| Research point cloud processing | 02/08/18 | 8 | 09/08/18 |
| Add ability to globally threshold point clouds | 02/08/18 | 8 | 09/08/18 |
| Add ability to de-noise point clouds | 02/08/18 | 8 | 09/08/18 |
| Add ability to downsample point clouds | 02/08/18 | 8 | 09/08/18 |
| Research point cloud registration | 02/08/18 | 43 | 13/09/18 |
| Add ability to register between point clouds | 02/08/18 | 15 | 16/08/18 |
| Add ability to use output of previous registration as input to next registration | 09/08/18 | 8 | 16/08/18 |
| Add ability to extract motion signal from registration output | 09/08/18 | 8 | 16/08/18 |
| Add ability to asses amount of movement between point clouds | 09/08/18 | 8 | 16/08/18 |
| Add ability to skip processing based on amount of movement between point clouds | 09/08/18 | 15 | 23/08/18 |
| Add ability to extract motion signal from movement between point clouds | 09/08/18 | 15 | 23/08/18 |
| Research point cloud tracking | 02/08/18 | 43 | 13/09/18 |
| Add ability to track a region of the point clouds | 16/08/18 | 29 | 13/09/18 |
| Add ability to extract motion signal from the tracked region of the point clouds | 16/08/18 | 8 | 23/08/18 |
| Add ability to change output settings and values for registration and motion signal extraction | 16/08/18 | 8 | 23/08/18 |
| Add ability to save values and output of registration and motion signal extraction to a text or binary file | 02/08/18 | 22 | 23/08/18 |
| Write final report | 13/08/18 | 32 | 13/09/18 |
| Edit final report | 06/09/18 | 8 | 13/09/18 |

*Figure 1: This illustration shows the data which was used to generate the final time plan Gantt chart.*

# Appendix F: Final Time Plan Gantt Chart

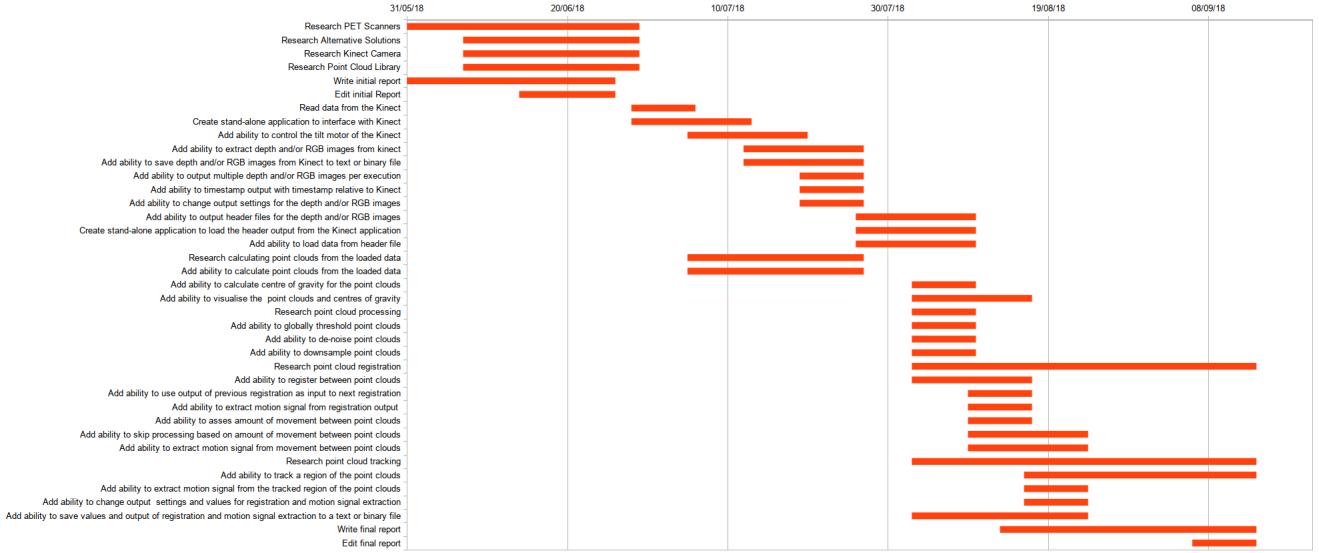


*Figure 2: This illustration shows the final time plan data as represented by a Gantt chart.*

# Appendix G: Initial Aims and Objectives

1. Port the STIR framework to Windows.
2. Read data from the Kinect Camera using the open source Kinect drivers.
3. *Create a stand-alone console application to interface with the Kinect camera.*
4. *Add the ability to save a point cloud from the Kinect camera to a file or data type.*
5. *Add the ability to output multiple point clouds per execution of the application.*
6. *Add the ability to timestamp the point clouds that are output by the Kinect camera.*
7. *Add the ability to clean the point clouds that are output by the Kinect camera.*
8. *Add the ability to register between the point clouds that are output by the Kinect camera.*
9. *Add the ability to extract a vector field or translation between two objects from the registered point clouds that were output from the Kinect camera.*
10. *Add the ability to remove any extraneous data from the point clouds that were output from the Kinect camera.*
11. *Add the ability to synchronise the timestamps contained within the point clouds that were output from the Kinect camera to the output from the PET scanner.*
12. *Add the ability to translate the origin of the space used by the Kinect camera to match the space used by the PET scanner.*
13. *Add the ability to apply the motion calculated from the output of the Kinect camera to the output of the PET scanner.*
14. Port the application to Windows.
15. Add the application to the windows version of the STIR framework as a library.

**Objective 1: Port the STIR framework to Windows**

The application should be able to be run on Windows, in order for that to be the case the STIR framework must be ported to Windows.

**Objective 2: Read data from the Kinect camera**

An application should be created which can read data from the Kinect camera in real time and visualise this data to the user.

The open source Kinect driver should be used for this application, this is because it allows for the development on and distribution to multiple operating systems.

### Objective 3: Create stand-alone console application to interface with Kinect

A stand-alone console application and library should be created which can either be launched directly as an executable program or can be compiled as a library and then accessed within another application.

To achieve this the application could be written as a library and then a wrapper application could be written to run it independently.

This is advantageous as it allows for more rapid prototyping of ideas without having to rely on a large codebase, which in itself could contain errors, without hindering the ease with which the application could be integrated into a larger framework.

This application should be able to receive point clouds from the Kinect camera.

### Objective 4: Add ability to save point cloud from Kinect to file or data type

The ability to save the data from one scan of the Kinect camera should be added to the application.

The data should be saved either to a file in storage or to a data type contained within the application.

A class or data type could be written that would be able to contain the data output from the Kinect camera, this class or data type should overload the streaming operators allowing it to be written either automatically to the console output or to a file.

### Objective 5: Add ability to output multiple point clouds per execution

The ability to save the data of multiple concurrent scans from the Kinect camera should be added to the application.

It is possible either to save each piece of data as they are read from the Kinect camera or to save all pieces of data at once after the scan has been complete.

Each new piece of data should be saved into a list or vector with the type of the class mentioned above. After execution has ended, a method could be called which would stream the entire list or vector either to the console output or to a file.

### Objective 6: Add ability to timestamp output

The ability to save the time that a scan occurred should be added to the application. This timestamp could be added as a data type to the class mentioned above, the timestamp should also be added to the overloaded streaming operators.

Timestamping the data from the Kinect camera is necessary to synchronize the Kinect camera output to the PET scanner output later.

### Objective 7: Add ability to clean point clouds

The ability to remove extraneous or outlying points from the data should be added to the application.

The process of cleaning the data can be either an automatic process or a manual one.

The PCL contains methods and algorithms that have been implemented and tested to clean point clouds.

### Objective 8: Add ability to register between point clouds

The ability to translate one set of points to another should be added to the application.

The registration process can be either rigid or non-rigid; this means that either the relationship between the points must remain the same for rigid registration or the relationship between the points can change for non-rigid registration.

The PCL contains methods and algorithms that have been implemented and tested to register point clouds.

### Objective 9: Add ability to extract vector field or translation from registered point clouds

The ability to extract the transformation from the registered data should be added to the application.

This could either be a vector field for non-rigid registration or a translation if rigid registration is used.

### Objective 10: Add ability to remove extraneous data from point cloud

The ability to remove structures that are not objects of interest should be added to the application.

In every set of data from the Kinect camera it is likely there will be structures that are of no importance to the functionality of the application, for instance the bezel of the PET scanner, these objects should be removed to cut down on processing power and memory usage.

A smaller data set is also easier to visualise clearly.

A point cloud could be acquired before scanning commences, then the difference between the initial point cloud and every other point cloud could be used to remove the bezel of the PET scanner and other extraneous data.

### Objective 11: Add ability to synchronise timestamp on point cloud to output from scanner

The ability to synchronise the output of the Kinect camera to the output of the PET scanner should be added to the application.

In order for each vector field or transformation extracted from the Kinect camera to be relevant it is important to know when this vector field or transformation was extracted in relation to the output of the PET scanner, this is to ensure that the correct vector field or transformation is applied to the correct data from the PET scanner.

One method to synchronise the output of the Kinect camera to the output of the PET scanner would be to use an Arduino to output a regular pulse of electricity and use this to set the clocks of both devices to the same rate.

### Objective 12: Add ability to translate camera space to scanner space

The ability to translate the output of the Kinect camera to the output of the PET scanner should be added to the application.

In order for each vector field or transformation extracted from the Kinect camera to be relevant the origin of both coordinate systems must be aligned, otherwise the same point in space could be represented in two different locations in both sets of data.

One method to translate the Kinect camera space to the PET scanner space would be to simultaneously scan a radiation point source with both the Kinect camera and the PET scanner and use this to calculate the transformation required.

### Objective 13: Add ability to apply output from application to output from scanner

The ability to translate the output of the PET scanner by the vector field or transformation of the data from Kinect camera should be added to the application.

This should apply the motion data gathered from the Kinect camera to the data from the PET scanner, this would effectively remove the motion recorded in the Kinect camera's data from the data of the PET scanner.

After this step has been applied, the output from PET scanner should appear to have a higher resolution and many motion related artefacts should be eliminated.

### Objective 14: Port application to Windows

The application should be able to run on a Windows machine.

This is to increase the potential user base for the application.

### Objective 15: Add application to the Windows version of STIR as library

The applications should be able to be added to the Windows version of the STIR framework as a library and integrated fully into its functionality.

The task list and time plan which have been determined from the above aims, hypothesis and objectives can be seen in the appendix below. (Appendix H:) (Appendix I:) (Appendix J:)

# Appendix H: Initial Task List

| # | Task Name | Description | Duration (weeks) |
|---|---|---|---|
| 1 | Research PET Scanners | Conduct background research specifically related to the operation of PET scanners. This research will aid in the writing of the initial report and the design of the application. | 3 |
| 2 | Research Alternative Solutions | Conduct background research specifically related to alternative solutions to the same problem that this project sets out to solve. This research will aid in the writing of the initial report and the design of the application. | 2 |
| 3 | Research Kinect Camera | Conduct background research specifically related to the operation of the Kinect Camera and its API. This research will aid in the writing of the initial report and the design of the application. | 2 |
| 4 | Research Point Cloud Library | Conduct background research related to the open source Point Cloud Library, specifically its ability to perform the cleaning of point clouds and the registration of point clouds. This research will aid in the writing of the initial report and the design of the application. | 2 |
| 5 | Write initial report | Write the initial report; this will involve writing the bulk of the content. | 4 |
| 6 | Edit initial Report | Edit the initial report, this will involve editing the content and ensuring it is presented to the best standard it could be. | 2 |

| | | | |
|---|---|---|---|
| 7 | Port STIR framework to Windows | The application should be able to be run on Windows, in order for that to be the case the STIR framework must be ported to Windows. | 2 |
| 8 | Read data from the Kinect | An application should be created which can read data from the Kinect camera in real time and visualise this data to the user. | 1 |
| 9 | Create stand-alone console application to interface with Kinect | A stand-alone console application should be created which can either be launched directly or can also be compiled as a library in order to use it as a part of a larger application. This application should be able to read data from the Kinect camera and communicate with the Kinect camera. | 1 |
| 10 | Add ability to save point cloud from Kinect to file or data type | The ability to save the data from one scan of the Kinect camera should be added to the application. The data should be saved either to a file in storage or to a data type contained within the application. | 1 |
| 11 | Add ability to output multiple point clouds per execution | The ability to save the data of multiple concurrent scans from the Kinect camera should be added to the application. It is possible either to save each piece of data as they are read from the Kinect camera or to save all pieces of data at once after the scan has been complete. | 1 |

| | | | |
|---|---|---|---|
| 12 | Add ability to timestamp output | The ability to save the time that a scan occurred should be added to the application. Timestamping the data from the Kinect camera is necessary to synchronize the Kinect camera output to the PET scanner output later. | 1 |
| 13 | Add ability to clean point clouds | The ability to remove extraneous or outlying points from the data should be added to the application. The process of cleaning the data can be either an automatic process or a manual one. | 2 |
| 14 | Add ability to register between point clouds | The ability to translate one set of points to another should be added to the application. At this stage, only the ability to recognise similar structures and match them between different sets of data is necessary. The registration process can be either rigid or non-rigid; this means that either the relationship between the points must remain the same for rigid registration or the relationship between the points can change for non-rigid registration. | 3 |
| 15 | Add ability to extract vector field or translation from registered point clouds | The ability to extract the transformation from the registered data should be added to the application. This could either be a vector field for non-rigid registration or a translation if rigid registration is used. | 3 |

| | | | |
|---|---|---|---|
| 16 | Add ability to remove extraneous data from point cloud | The ability to remove structures that are not objects of interest should be added to the application. In every set of data from the Kinect camera it is likely there will be structures that are of no importance to the functionality of the application, for instance the bezel of the PET scanner, these objects should be removed to cut down on processing power and memory usage. A smaller data set is also easier to visualise clearly. | 2 |
| 17 | Add ability to synchronise timestamp on point cloud to output from scanner | The ability to synchronise the output of the Kinect camera to the output of the PET scanner should be added to the application. In order for each vector field or translation extracted from the Kinect camera to be relevant it is important to know when this vector field or translation was extracted in relation to the output of the PET scanner, this is to ensure that the correct vector field or translation is applied to the correct data from the PET scanner. | 2 |
| 18 | Add ability to translate camera space to scanner space | The ability to translate the output of the Kinect camera to the output of the PET scanner should be added to the application. In order for each vector field or translation extracted from the Kinect camera to be relevant the origin of both coordinate systems must be aligned, otherwise the same point in space could be represented in two different locations in both sets of data. | 2 |

| | | | |
|---|---|---|---|
| 19 | Add ability to apply output from application to output from scanner | The ability to translate the output of the PET scanner by the vector field or translation of the data from Kinect camera should be added to the application. This should apply the motion data gathered from the Kinect camera to the data from the PET scanner, this would effectively remove the motion recorded in the Kinect camera's data from the data of the PET scanner. After this step has been applied, the output from PET scanner should appear to have a higher resolution and many motion related artefacts should be eliminated. | 3 |
| 20 | Write final report | Write the final report; this will involve writing the bulk of the content. | 8 |
| 21 | Edit final report | Edit the final report, this will involve editing the content and ensuring it is presented to the best standard it could be. | 3 |
| 22 | Port application to Windows | The application should be able to run on a Windows machine. This is to increase the potential user base for the application but also to ensure that the application is compatible with the STIR framework. | 2 |
| 23 | Add application to the Windows version of STIR as library | The applications should be able to be added to the Windows version of the STIR framework as a library and integrated fully into its functionality. | 2 |

*Table 3: This table shows the initial list of tasks that must be completed to finish the project. These tasks are then expanded upon by providing a short description of each task and an estimated duration in weeks.*

# Appendix I: Initial Time Plan Table

| | | | |
|---|---|---|---|
| Research PET Scanners | 31/05/18 | 25 | 24/06/18 |
| Research Alternative Solutions | 11/06/18 | 14 | 24/06/18 |
| Research Kinect Camera | 11/06/18 | 14 | 24/06/18 |
| Research Point Cloud Library | 11/06/18 | 14 | 24/06/18 |
| Write initial report | 31/05/18 | 26 | 25/06/18 |
| Edit initial Report | 11/06/18 | 15 | 25/06/18 |
| Port STIR framework to Windows | 11/06/18 | 14 | 24/06/18 |
| Read data from the Kinect | 11/06/18 | 7 | 17/06/18 |
| Create stand-alone console application to interface with Kinect | 25/06/18 | 7 | 01/07/18 |
| Add ability to save point cloud from Kinect to file or data type | 25/06/18 | 7 | 01/07/18 |
| Add ability to output multiple point clouds per execution | 25/06/18 | 7 | 01/07/18 |
| Add ability to timestamp output | 25/06/18 | 7 | 01/07/18 |
| Add ability to clean point clouds | 02/07/18 | 14 | 15/07/18 |
| Add ability to register between point clouds | 02/07/18 | 21 | 22/07/18 |
| Add ability to extract vector field or translation from registered point clouds | 09/07/18 | 21 | 29/07/18 |
| Add ability to remove extraneous data from point cloud | 16/07/18 | 14 | 29/07/18 |
| Add ability to synchronise timestamp on point cloud to output from scanner | 13/08/18 | 14 | 26/08/18 |
| Add ability to translate camera space to scanner space | 13/08/18 | 14 | 26/08/18 |
| Add ability to apply output from application to output from scanner | 20/08/18 | 21 | 09/09/18 |
| Write final report | 30/07/18 | 54 | 21/09/18 |

*Figure 3: This illustration shows the data which was used to generate the initial time plan Gantt chart.*

# Appendix J: Initial Time Plan Gantt Chart

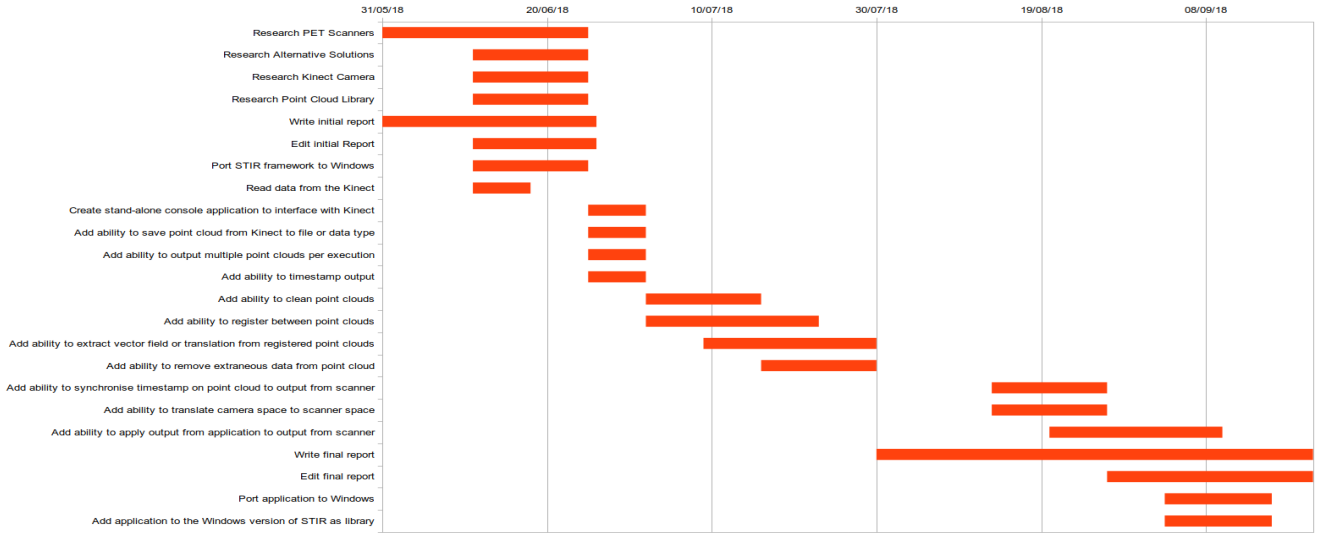


*Figure 4: This illustration shows the initial time plan data as represented by a Gantt chart.*

# Appendix K: Microsoft Kinect Camera Version Comparison

| Scanner | Scanner Type | Minimum Range (mm) | Maximum Range (mm) | Horizontal Resolution (px) | Vertical Resolution (px) | Scanning Rate (Hz) | Field of View (deg) |
|---|---|---|---|---|---|---|---|
| Kinect V1 | Structured Light | 400 | 4000 | 640 | 480 | 30 | 60 |
| Kinect V2 | Time of Flight | 500 | 8000 | 512 | 424 | 30 | 70 |

*Illustration 128: This illustration shows a comparison between the Kinect camera version one and the Kinect camera version two.*

# Appendix L: Point Cloud Data Example

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z
SIZE 4 4 4
TYPE F F F
COUNT 1 1 1
WIDTH 307200
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 307200
DATA ascii
-55.439999 -73.919998 100
-55.209 -73.919998 100
-54.977997 -73.919998 100
-54.746998 -73.919998 100
-54.515999 -73.919998 100
-54.285 -73.919998 100
-54.053997 -73.919998 100
-53.822998 -73.919998 100
-53.591999 -73.919998 100
-53.360996 -73.919998 100
-53.129997 -73.919998 100
```

# Appendix M: Kinect Point Cloud Library Processing Example

kpclp_header_version=0.1

data_type=u

data_size=16

data_dimensions=1

data_resolution=640,480

data_path=/home/nikos/Documents/KinectPointCloudTest-output/depth_test_0.bin

epoch_timestamp=0

kinect_timestamp=0

kpclp_header_status=end

# Appendix N: Output Example

Registration Type: ICP

Registration Style: Continuous

Movement Type: Distance

Movement Distance: 1.000000

Translation Estimation: Automatic

Tracking Type: Complex

Threshold: 1000.000000

Offset: -10

Focal Length: 0.002100

Smoothing Size: 1

Smoothing Deviation: 1.000000

Filter (xyz): 1.000000, 1.000000, 1.000000

Transformation Epsilon: 0.010000

Iterations: 100

Rotation Guess: 0.000000

Translation Guess (xyz): 0.000000, 0.000000, 0.000000

Signal Magnitude: 50.000000

Signal (xyz): 0.000000, 0.000000, 0.000000

File Paths 0 1: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_1.pcd

File Paths 0 2: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_2.pcd

File Paths 0 3: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_3.pcd

File Paths 0 4: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_4.pcd

File Paths 0 5: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_5.pcd

File Paths 0 6: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_6.pcd

File Paths 0 7: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_7.pcd

File Paths 0 8: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_8.pcd

File Paths 0 9: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_9.pcd

File Paths 0 10: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_10.pcd

File Paths 0 11: /home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_0.pcd,/home/nikos/Documents/MSc-Dissertation/Report/Misc/Test Data/Simple Plane Test/point_cloud_test_11.pcd

Registration 0 1:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 1: 38.666550,-16.495033,104.954422,1.000000

Centroid Differences 0 1: 20.167114,2.012743,4.954422,0.000000

Signal 0 1:

Signal Mean Square Error 0 1: 0.157254

Signal Transform 0 1: 0.994428,-0.053583,0.090784,4.952641,0.061812,0.993988,-0.090409,8.978785,-0.085393,0.095517,0.991756,9.531178,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 1: 0.572870

Transform 0 1: 0.994420,-0.053724,0.090784,4.955229,0.061954,0.993979,-0.090409,8.975059,-0.085380,0.095529,0.991756,9.531173,0.000000,0.000000,0.000000,1.000000

Registration 0 2:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 2: 56.790272,-13.738134,109.035439,1.000000

Centroid Differences 0 2: 38.290836,4.769642,9.035439,0.000000

Signal 0 2:

Signal Mean Square Error 0 2: 0.146404

Signal Transform 0 2: 0.974315,-0.134243,0.180819,13.979696,0.166632,0.969855,-0.177839,18.000423,-0.151493,0.203400,0.967308,19.491947,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 2: 0.530187

Transform 0 2: 0.974319,-0.134210,0.180819,13.978579,0.166599,0.969861,-0.177839,18.001505,-0.151500,0.203396,0.967308,19.493010,0.000000,0.000000,0.000000,1.000000

Registration 0 3:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 3: 74.318802,-10.085920,112.823669,1.000000

Centroid Differences 0 3: 55.819366,8.421856,12.823669,0.000000

Signal 0 3:

Signal Mean Square Error 0 3: 0.128189

Signal Transform 0 3: 0.938767,-0.214880,0.269361,22.216324,0.285604,0.922581,-0.259407,26.402935,-0.192764,0.320450,0.927452,30.338066,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 3: 0.456768

Transform 0 3: 0.938760,-0.214911,0.269361,22.216936,0.285635,0.922572,-0.259407,26.400969,-0.192754,0.320457,0.927452,30.337282,0.000000,0.000000,0.000000,1.000000

Registration 0 4:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 4: 90.381516,-5.948565,117.046921,1.000000

Centroid Differences 0 4: 71.882080,12.559212,17.046921,0.000000

Signal 0 4:

Signal Mean Square Error 0 4: 0.047984

Signal Transform 0 4: 0.888279,-0.290635,0.355679,29.750956,0.410867,0.848943,-0.332423,33.872314,-0.205336,0.441417,0.873505,42.404846,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 4: 0.330061

Transform 0 4: 0.888290,-0.290601,0.355679,29.753340,0.410834,0.848959,-0.332422,33.873562,-0.205353,0.441409,0.873505,42.404240,0.000000,0.000000,0.000000,1.000000

Registration 0 5:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 5: 104.764709,-1.728076,122.362762,1.000000

Centroid Differences 0 5: 86.265274,16.779699,22.362762,0.000000

Signal 0 5:

Signal Mean Square Error 0 5: 0.078565

Signal Transform 0 5: 0.825210,-0.355354,0.439060,36.275799,0.532404,0.748971,-0.394480,39.849228,-0.188661,0.559280,0.807244,56.050114,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 5: 0.318931

Transform 0 5: 0.825187,-0.355406,0.439060,36.276829,0.532451,0.748937,-0.394480,39.847755,-0.188625,0.559292,0.807243,56.048771,0.000000,0.000000,0.000000,1.000000

Registration 0 6:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 6: 117.477661,1.804959,129.163528,1.000000

Centroid Differences 0 6: 98.978226,20.312735,29.163528,0.000000

Signal 0 6:

Signal Mean Square Error 0 6: 0.120766

Signal Transform 0 6: 0.751815,-0.406978,0.518813,42.001377,0.643682,0.623684,-0.443533,43.743069,-0.143067,0.667399,0.730852,71.361633,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 6: 0.331682

Transform 0 6: 0.751829,-0.406953,0.518813,42.003330,0.643661,0.623706,-0.443533,43.742077,-0.143089,0.667395,0.730852,71.359337,0.000000,0.000000,0.000000,1.000000

Registration 0 7:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 7: 128.754944,3.957525,137.611176,1.000000

Centroid Differences 0 7: 110.255508,22.465302,37.611176,0.000000

Signal 0 7:

Signal Mean Square Error 0 7: 0.119605

Signal Transform 0 7: 0.669766,-0.445280,0.594286,47.135029,0.739380,0.474244,-0.477967,44.754288,-0.069008,0.759518,0.646851,88.141838,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 7: 0.289452

Transform 0 7: 0.669764,-0.445283,0.594286,47.134510,0.739382,0.474241,-0.477967,44.754665,-0.069004,0.759518,0.646851,88.142296,0.000000,0.000000,0.000000,1.000000

Registration 0 8:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 8: 138.809784,3.888212,147.622665,1.000000

Centroid Differences 0 8: 120.310349,22.395988,47.622665,0.000000

Signal 0 8:

Signal Mean Square Error 0 8: 0.118274

Signal Transform 0 8: 0.583009,-0.467035,0.664852,51.910713,0.812006,0.306662,-0.496643,42.327339,0.028062,0.829394,0.558010,106.153511,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 8: 0.264903

Transform 0 8: 0.582992,-0.467055,0.664852,51.910461,0.812017,0.306633,-0.496643,42.326672,0.028091,0.829393,0.558010,106.153252,0.000000,0.000000,0.000000,1.000000

Registration 0 9:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 9: 147.970505,0.709593,158.736450,1.000000

Centroid Differences 0 9: 129.471069,19.217369,58.736450,0.000000

Signal 0 9:

Signal Mean Square Error 0 9: 0.051814

Signal Transform 0 9: 0.493999,-0.472473,0.729928,56.702309,0.857492,0.125758,-0.498946,35.620255,0.143936,0.872365,0.467254,124.560333,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 9: 0.193217

Transform 0 9: 0.493787,-0.472693,0.729928,56.704136,0.857548,0.125374,-0.498946,35.632458,0.144327,0.872300,0.467254,124.570770,0.000000,0.000000,0.000000,1.000000

Registration 0 10:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 10: 156.813873,-6.017352,170.234604,1.000000

Centroid Differences 0 10: 138.314438,12.490425,70.234604,0.000000

Signal 0 10:

Signal Mean Square Error 0 10: 0.050544

Signal Transform 0 10: 0.404011,-0.462994,0.788971,61.951015,0.872240,-0.064985,-0.484796,24.572380,0.275716,0.884017,0.377576,142.867340,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 10: 0.166983

Transform 0 10: 0.404016,-0.462990,0.788971,61.952599,0.872241,-0.064976,-0.484796,24.575504,0.275706,0.884020,0.377576,142.868103,0.000000,0.000000,0.000000,1.000000

Registration 0 11:

Source Centroid 0: 18.499435,-18.507776,100.000000,1.000000

Target Centroid 11: 165.830917,-16.575571,181.284714,1.000000

Centroid Differences 0 11: 147.331482,1.932205,81.284714,0.000000

Signal 0 11:

Signal Mean Square Error 0 11: 0.128292

Signal Transform 0 11: 0.317482,-0.437208,0.841496,68.201012,0.853267,-0.255480,-0.454665,8.752795,0.413751,0.862353,0.291934,159.580734,0.000000,0.000000,0.000000,1.000000

Mean Square Error 0 11: 0.246621

Transform 0 11: 0.317453,-0.437228,0.841496,68.203056,0.853250,-0.255538,-0.454664,8.752989,0.413809,0.862325,0.291934,159.579651,0.000000,0.000000,0.000000,1.000000

# References

*3D Viewing: the Pinhole Camera Model (A Virtual Pinhole Camera Model)* (no date). Available at: https://www.scratchapixel.com/lessons/3d-basic-rendering/3d-viewing-pinhole-camera/virtual-pinhole-camera-model (Accessed: 10 September 2018).

Alexa, M. *et al.* (2003) 'Computing and rendering point set surfaces', *IEEE Transactions on Visualization and Computer Graphics*, 9(1), pp. 3–15. doi: 10.1109/TVCG.2003.1175093.

*Alexander Charles Whitehead / KinectLibraries · GitLab* (no date). Available at: https://gitlab.com/ALEXJAZZ008008/KinectLibraries (Accessed: 13 September 2018).

*Alexander Charles Whitehead / KinectPointCloudExperiment · GitLab* (no date). Available at: https://gitlab.com/ALEXJAZZ008008/KinectPointCloudExperiment (Accessed: 13 September 2018).

*Alexander Charles Whitehead / KinectPointCloudTest · GitLab* (no date). Available at: https://gitlab.com/ALEXJAZZ008008/KinectPointCloudTest (Accessed: 13 September 2018).

*Alexander Charles Whitehead / MSc-Dissertation · GitLab* (no date). Available at: https://gitlab.com/ALEXJAZZ008008/MSc-Dissertation (Accessed: 13 September 2018).

*Alexander Charles Whitehead / PointCloudProcessing · GitLab* (no date). Available at: https://gitlab.com/ALEXJAZZ008008/PointCloudProcessing (Accessed: 13 September 2018).

*ALEXJAZZ008008/KinectLibraries* (no date). Available at: https://github.com/ALEXJAZZ008008/KinectLibraries (Accessed: 13 September 2018).

*ALEXJAZZ008008/KinectPointCloudExperiment* (no date). Available at: https://github.com/ALEXJAZZ008008/KinectPointCloudExperiment (Accessed: 13 September 2018).

*ALEXJAZZ008008/KinectPointCloudTest* (no date). Available at: https://github.com/ALEXJAZZ008008/KinectPointCloudTest (Accessed: 13 September 2018).

*ALEXJAZZ008008/MSc-Dissertation* (no date). Available at: https://github.com/ALEXJAZZ008008/MSc-Dissertation (Accessed: 13 September 2018).

*ALEXJAZZ008008/PointCloudProcessing* (no date). Available at: https://github.com/ALEXJAZZ008008/PointCloudProcessing (Accessed: 13 September 2018).

Bayardo, R. J. *et al.* (2004) 'An evaluation of binary xml encoding optimizations for fast stream based xml processing', in *Proceedings of the 13th conference on World Wide Web - WWW '04*. New York, New York, USA: ACM Press, p. 345. doi: 10.1145/988672.988719.

Bertolli, O. (2018) *Data-Driven methods for respiratory signal detection in Positron Emission Tomography*. University College London. Available at: https://liveuclac-my.sharepoint.com/personal/rmhathi_ucl_ac_uk/_layouts/15/onedrive.aspx?id=%252Fpersonal%252Frmhathi%255Fucl%255Fac%255Fuk%252FDocuments%252FINM%252FINM%252Dshared%252Fpublications%252Fthesis%252FBertolli%252D2018%252DPhD%252Dthesis%252Ep (Accessed: 21 June 2018).

Besl, P. J. and McKay, N. D. (1992) 'A method for registration of 3-D shapes', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), pp. 239–256. doi: 10.1109/34.121791.

Beyer, T. *et al.* (2003) 'Dual-modality PET/CT imaging: the effect of respiratory motion on combined image quality in clinical oncology', *European Journal of Nuclear Medicine and Molecular Imaging*. Springer-Verlag, 30(4), pp. 588–596. doi: 10.1007/s00259-002-1097-6.

Bousse, A. *et al.* (2016) 'Maximum-Likelihood Joint Image Reconstruction/Motion Estimation in Attenuation-Corrected Respiratory Gated PET/CT Using a Single Attenuation Map', *IEEE Transactions on Medical Imaging*, 35(1), pp. 217–228. doi: 10.1109/TMI.2015.2464156.

Bousse, A. *et al.* (2017) 'Evaluation of a direct motion estimation/correction method in respiratory-gated PET/MRI with motion-adjusted attenuation', *Medical Physics*. Wiley-Blackwell, 44(6), pp. 2379–2390. doi: 10.1002/mp.12253.

Burger, C. *et al.* (2002) 'PET attenuation coefficients from CT images: experimental evaluation of the transformation of CT into PET 511-keV attenuation coefficients', *European Journal of Nuclear Medicine and Molecular Imaging*. Springer-Verlag, 29(7), pp. 922–927. doi: 10.1007/s00259-002-0796-3.

Burgos, N. *et al.* (2014) 'Attenuation Correction Synthesis for Hybrid PET-MR Scanners: Application to Brain Studies', *IEEE Transactions on Medical Imaging*, 33(12), pp. 2332–2341. doi: 10.1109/TMI.2014.2340135.

*C++ GUI Programming with Qt4 - Jasmin Blanchette, Mark Summerfield - Google Books* (no date). Available at: https://books.google.co.uk/books?id=ia-smJ2_ClsC&pg=PT21&lpg=PT21&dq=qt+popularity&source=bl&ots=O3jNJEhdiF&sig=

mfLpilIziePix15LA9g2_pHxkXA&hl=en&sa=X&ved=2ahUKEwju6L3wkLfdAhVHI8AKHTIMAcsQ6AEwBHoECAcQAQ#v=onepage&q=qt popularity&f=false (Accessed: 13 September 2018).

Chan, C. *et al.* (2018) 'Non-Rigid Event-by-Event Continuous Respiratory Motion Compensated List-Mode Reconstruction for PET', *IEEE Transactions on Medical Imaging*, 37(2), pp. 504–515. doi: 10.1109/TMI.2017.2761756.

*Color supported generalized-ICP* (no date). Available at: https://www.computer.org/csdl/proceedings/visapp/2014/8133/03/07295135-abs.html (Accessed: 10 September 2018).

'Comparison of different methods for data-driven respiratory gating of PET data' (2013) in *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC)*. IEEE, pp. 1–4. doi: 10.1109/NSSMIC.2013.6829055.

Conti, M. (2011) 'Focus on time-of-flight PET: the benefits of improved time resolution', *European Journal of Nuclear Medicine and Molecular Imaging*. Springer-Verlag, 38(6), pp. 1147–1157. doi: 10.1007/s00259-010-1711-y.

*Documentation - Point Cloud Library (PCL)* (no date a). Available at: http://pointclouds.org/documentation/tutorials/tracking.php (Accessed: 10 September 2018).

*Documentation - Point Cloud Library (PCL)* (no date b). Available at: http://pointclouds.org/documentation/tutorials/registration_api.php (Accessed: 10 September 2018).

*Documentation - Point Cloud Library (PCL)* (no date c). Available at: http://pointclouds.org/documentation/tutorials/statistical_outlier.php (Accessed: 10 September 2018).

*Documentation - Point Cloud Library (PCL)* (no date d). Available at: http://pointclouds.org/documentation/tutorials/walkthrough.php (Accessed: 10 September 2018).

*Documentation - Point Cloud Library (PCL)* (no date e). Available at: http://pointclouds.org/documentation/tutorials/conditional_euclidean_clustering.php (Accessed: 10 September 2018).

Drzezga, A. *et al.* (2012) 'First clinical experience with integrated whole-body PET/MR: comparison to PET/CT in patients with oncologic diagnoses.', *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*. Society of Nuclear Medicine, 53(6), pp. 845–55. doi: 10.2967/jnumed.111.098608.

Eng and Eirik (1994) *Linux journal.*, *Linux Journal*. Robert F. Young. Available at: https://dl.acm.org/citation.cfm?id=326466 (Accessed: 13 September 2018).

*FAQ - OpenKinect* (no date). Available at: https://openkinect.org/wiki/FAQ (Accessed: 13 September 2018).

Fischler, M. A. and Bolles, R. C. (1981) 'Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography', *Communications of the ACM*. ACM, 24(6), pp. 381–395. doi: 10.1145/358669.358692.

Geladi, P. and Kowalski, B. R. (1986) 'Partial least-squares regression: a tutorial', *Analytica Chimica Acta*. Elsevier, 185, pp. 1–17. doi: 10.1016/0003-2670(86)80028-9.

*Heart-direct-vs-iterative-reconstruction.png (PNG Image, 516 × 256 pixels)* (no date). Available at: https://upload.wikimedia.org/wikipedia/commons/5/54/Heart-direct-vs-iterative-reconstruction.png (Accessed: 10 September 2018).

Hudson, H. M. and Larkin, R. S. (1994) 'Accelerated image reconstruction using ordered subsets of projection data', *IEEE Transactions on Medical Imaging*, 13(4), pp. 601–609. doi: 10.1109/42.363108.

*Imaging Information - OpenKinect* (no date). Available at: https://openkinect.org/wiki/Imaging_Information (Accessed: 12 September 2018).

Jain, A. K., Murty, M. N. and Flynn, P. J. (1999) 'Data clustering: a review', *ACM Computing Surveys*. ACM, 31(3), pp. 264–323. doi: 10.1145/331499.331504.

Kadrmas, D. J. (2004) 'LOR-OSEM: statistical PET reconstruction from raw line-of-response histograms', *Physics in Medicine and Biology*. IOP Publishing, 49(20), pp. 4731–4744. doi: 10.1088/0031-9155/49/20/005.

Khoshelham, K. and Elberink, S. O. (2012) 'Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications', *Sensors*. Molecular Diversity Preservation International, 12(2), pp. 1437–1454. doi: 10.3390/s120201437.

Lau, D. (2013) *Gamasutra: Daniel Lau's Blog - The Science Behind Kinects or Kinect 1.0 versus 2.0*. Available at: https://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php (Accessed: 24 June 2018).

Levinson, N. (1946) 'The Wiener (Root Mean Square) Error Criterion in Filter Design and Prediction', *Journal of Mathematics and Physics*. Wiley/Blackwell (10.1111), 25(1–4), pp. 261–278. doi: 10.1002/sapm1946251261.

Lindner, M., Schiller, I. and Koch, R. (2010) 'Time-of-Flight sensor calibration for accurate range sensing', *Computer Vision and Image Understanding*. Academic Press, 114(12), pp. 1318–1328. doi: 10.1016/J.CVIU.2009.11.002.

Liu, C. *et al.* (2009) 'The impact of respiratory motion on tumor quantification and delineation in static PET/CT imaging', *Physics in Medicine and Biology*. IOP Publishing, 54(24), pp. 7345–7362. doi: 10.1088/0031-9155/54/24/007.

Lowe, V. J. *et al.* (1995) ‘Optimum scanning protocol for FDG-PET evaluation of pulmonary malignancy.’, *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*, 36(5), pp. 883–7. Available at: http://www.ncbi.nlm.nih.gov/pubmed/7738668 (Accessed: 12 September 2018).

*Mabel Lowman Art: long exposure photography* (no date). Available at: http://mabellowman.blogspot.com/2012/10/long-exposure-photography.html (Accessed: 10 September 2018).

Magnusson, M. (2009) *The Three-Dimensional Normal-Distributions Transform an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. Örebro universitet.

Manber, R. *et al.* (2015) ‘Practical PET Respiratory Motion Correction in Clinical PET/MR.’, *Journal of nuclear medicine : official publication, Society of Nuclear Medicine*. Society of Nuclear Medicine, 56(6), pp. 890–6. doi: 10.2967/jnumed.114.151779.

Manber, R. *et al.* (2016) ‘Joint PET-MR respiratory motion models for clinical PET motion correction.’, *Physics in medicine and biology*, 61(17), pp. 6515–30. doi: 10.1088/0031-9155/61/17/6515.

McClelland, J. R. *et al.* (2017) ‘A generalized framework unifying image registration and respiratory motion models and incorporating image reconstruction, for partial image data or full images’, *Physics in Medicine and Biology*. IOP Publishing, 62(11), pp. 4273–4292. doi: 10.1088/1361-6560/aa6070.

Micallef, J. (1987) ‘Encapsulation, Reusability and Extensibility in Object-Oriented Programming Languages’. doi: 10.7916/D8TT4ZZD.

Miranda, A. *et al.* (2017) ‘Markerless rat head motion tracking using structured light for brain PET imaging of unrestrained awake small animals’, *Physics in Medicine and Biology*. IOP Publishing, 62(5), pp. 1744–1758. doi: 10.1088/1361-6560/aa5a46.

*Moving_Least_Squares2.png (PNG Image, 560 × 420 pixels)* (no date). Available at: https://upload.wikimedia.org/wikipedia/commons/f/f3/Moving_Least_Squares2.png (Accessed: 10 September 2018).

Nandi, A. (2013) *How do PET scans differentiate depth along a cross-section? - Quora*. Available at: https://www.quora.com/How-do-PET-scans-differentiate-depth-along-a-cross-section (Accessed: 22 June 2018).

Natarajan, P. *et al.* (2012) ‘Tumor detection using threshold operation in MRI brain images’, in *2012 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, pp. 1–4. doi: 10.1109/ICCIC.2012.6510299.

*New PET MRI scanner is first of its kind in UK* (2012). Available at: http://www.uclh.nhs.uk/Research/BRC/News/Pages/NewPETMRIscannerisfirstofitskindinUK.aspx (Accessed: 22 June 2018).

*NikEfth/PPDD: SAvVy - Stir dAta Viewer* (no date). Available at: https://github.com/NikEfth/PPDD (Accessed: 15 July 2018).

*NikEfth/SAvVy: SAvVy - Stir dAta Viewer* (2018). Available at: https://github.com/NikEfth/SAvVy (Accessed: 25 June 2018).

Noonan, P. J. *et al.* (2015) 'Repurposing the Microsoft Kinect for Windows v2 for external head motion tracking for brain PET', *Physics in Medicine and Biology*. IOP Publishing, 60(22), pp. 8753–8766. doi: 10.1088/0031-9155/60/22/8753.

*OpenKinect/libfreenect: Drivers and libraries for the Xbox Kinect device on Windows, Linux, and OS X* (2018). Available at: https://github.com/OpenKinect/libfreenect (Accessed: 25 June 2018).

*OpenKinect/libfreenect2: Open source drivers for the Kinect for Windows v2 device* (2018). Available at: https://github.com/OpenKinect/libfreenect2 (Accessed: 25 June 2018).

*PCL Developers Blog* (2015). Available at: http://pointclouds.org/blog/ (Accessed: 25 June 2018).

*pcl downsample voxel grid - YouTube* (no date). Available at: https://www.youtube.com/watch?v=IKkztrLwRco (Accessed: 10 September 2018).

Picard, Y. and Thompson, C. J. (1997) 'Motion correction of PET images using multiple acquisition frames', *IEEE Transactions on Medical Imaging*, 16(2), pp. 137–144. doi: 10.1109/42.563659.

*Point Cloud Processing / Data Management | H2H Associates* (2018). Available at: http://h2hassociates.com/services/point-cloud-processing-data-management/ (Accessed: 23 June 2018).

*PointCloudLibrary/pcl: Point Cloud Library (PCL)* (2018). Available at: https://github.com/PointCloudLibrary/pcl (Accessed: 25 June 2018).

Poulin, M. (2014) *LEDs and sensing technology enable new ranging applications (MAGAZINE) - LEDs*. Available at: https://www.ledsmagazine.com/articles/print/volume-11/issue-6/features/developer-forum/leds-and-sensing-technology-enable-new-ranging-applications.html (Accessed: 24 June 2018).

'Prefilter collimator for PET gamma camera' (1997). Available at: https://patents.google.com/patent/US5751000A/en (Accessed: 12 September 2018).

*Qt, GTK+ - Explore - Google Trends* (no date). Available at: https://trends.google.com/trends/explore?date=all&q=%2Fm%2F069fr,%2Fm%2F0chq_ (Accessed: 10 September 2018).

*qt/qtbase: Qt Base (Core, Gui, Widgets, Network, ...)* (2018). Available at: https://github.com/qt/qtbase (Accessed: 25 June 2018).

*RANSAC_Inliers_and_Outliers.png (PNG Image, 986 × 460 pixels)* (no date). Available at: https://upload.wikimedia.org/wikipedia/commons/b/b7/RANSAC_Inliers_and_Outliers.png (Accessed: 10 September 2018).

Ray, B. *et al.* (2014) 'A large scale study of programming languages and code quality in github', in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*. New York, New York, USA: ACM Press, pp. 155–165. doi: 10.1145/2635868.2635922.

Ren, S. *et al.* (2017) 'Data-driven event-by-event respiratory motion correction using TOF PET list-mode centroid of distribution', *Physics in Medicine and Biology*. IOP Publishing, 62(12), pp. 4741–4755. doi: 10.1088/1361-6560/aa700c.

Rowe, C. C. *et al.* (2008) 'Imaging of amyloid β in Alzheimer's disease with 18F-BAY94-9172, a novel PET tracer: proof of mechanism', *The Lancet Neurology*. Elsevier, 7(2), pp. 129–135. doi: 10.1016/S1474-4422(08)70001-2.

*Running GTK+ Applications: GTK+ 3 Reference Manual* (no date). Available at: https://developer.gnome.org/gtk3/stable/gtk-running.html (Accessed: 13 September 2018).

Rusu, R. B. and Cousins, S. (2011) '3D is here: Point Cloud Library (PCL)', in *2011 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1–4. doi: 10.1109/ICRA.2011.5980567.

Sagara, Y. *et al.* (2010) 'Abdominal CT: Comparison of Low-Dose CT With Adaptive Statistical Iterative Reconstruction and Routine-Dose CT With Filtered Back Projection in 53 Patients', *American Journal of Roentgenology*. American Roentgen Ray Society, 195(3), pp. 713–719. doi: 10.2214/AJR.09.2989.

Smeenk, R. (2014) *Kinect V1 and Kinect V2 fields of view compared - Roland Smeenk*. Available at: https://smeenk.com/kinect-field-of-view-comparison/ (Accessed: 23 June 2018).

Staranowicz, A. N. *et al.* (2015) 'Practical and accurate calibration of RGB-D cameras using spheres', *Computer Vision and Image Understanding*. Academic Press, 137, pp. 102–114. doi: 10.1016/J.CVIU.2015.03.013.

*STL File Format (3D Printing) - Simply Explained* | *All3DP* (no date). Available at: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/ (Accessed: 13 September 2018).

*The PLY Polygon File Format* (no date). Available at: https://web.archive.org/web/20161204152348/http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/ply.html (Accessed: 8 September 2018).

Thielemans, K. *et al.* (2012) 'STIR: software for tomographic image reconstruction release 2', *Physics in Medicine and Biology*, 57(4), pp. 867–883. doi: 10.1088/0031-9155/57/4/867.

*Tomographic Reconstruction in Nuclear Medicine* | *Radiology Key* (no date). Available at: https://radiologykey.com/tomographic-reconstruction-in-nuclear-medicine/ (Accessed: 10 September 2018).

Townsend, D. W., Beyer, T. and Blodgett, T. M. (2003) 'PET/CT scanners: A hardware approach to image fusion', *Seminars in Nuclear Medicine*. W.B. Saunders, 33(3), pp. 193–204. doi: 10.1053/SNUC.2003.127314.

*UCL/STIR: Software for Tomographic Image Reconstruction* (2018). Available at: https://github.com/UCL/STIR (Accessed: 25 June 2018).

Zhang, Z. (2012) ‘Microsoft Kinect Sensor and Its Effect’, *IEEE Multimedia*, 19(2), pp. 4–10. doi: 10.1109/MMUL.2012.24.