University of Nairobi

School of Engineering

Building a QGIS Helper Application to Overcome the Challenges of Cassini to UTM Coordinate System Conversions in Kenya

BY

Ajanga Dissent Ingati

F56/81982/2015

A Project submitted in partial fulfillment for the Degree of Master of Science in Geographical Information Systems, in the Department of Geospatial & Space Technology of the University of Nairobi

October 2017

Declaration

I, Ajanga Dissent Ingati, hereby declare that this project is my original work. To the best of my knowledge, the work presented here has not been presented for a degree in any other Institution of Higher Learning.

…….	……	……..……
Name of student	Signature	Date

This project has been submitted for examination with my approval as university supervisor.

…….	……	……..……
Name of Supervisor	Signature	Date

Dedication

I dedicate this work to my family for their support throughout the entire process of pursuing this degree and to the completion of the project. Your encouragement has been instrumental in ensuring the success of this project; may God truly bless you.

Acknowledgement

I would like to show my heartfelt appreciation to all the people who saw me through the process of preparing this report. These thanks go to all those who provided technical support, talked through the issues I studied, read, wrote, gave comments, and also those who assisted in editing and proof reading of this project report before submission. I would like to thank Steve Firsake for his technical support through Python Programing. Special thanks also go to Dr.-Ing. Musyoka for his technical advice and support for the entire project.

Table of contents

List of Tables

List of Figures

List of Abbreviations

DOS	Directorate of Overseas Surveys
GIS	Geographical Information Systems
UTM	Universal Transverse Mercator
GUI	Graphic User Interface
IDE	Integrated Development Environment
ESRI	Environmental Systems Research Institute
BSD	Berkeley Software Distribution
GDAL	Geospatial Data Abstraction Library
CRS	Coordinate Reference System
SQL	Structured Query Language
PHP	Hypertext Pre-processor
VBA	Visual Basic

Abstract

Three major coordinate systems have been in use in Kenya. They include; Cassini-Soldner, East African War system and UTM coordinate systems. The existence of these multiple systems have triggered the necessity for regular conversions particularly from Cassini to UTM. This is mainly because in Kenya, cadastral surveys are based on Cassini while topographic maps are based on UTM. Thus, it becomes a challenge for users to merge data sets in these two systems.

The core objective of this study was to build a helper application to overcome the challenges of Cassini and UTM coordinate system conversions. The methodology used involved carrying out a desk study, presentation and discussion of results from the study and the design of a QGIS helper application that converts coordinates between Cassini and UTM.

The desk study involved carrying out a review of existing information on methods currently applied in Kenya to facilitate the conversions. This was aimed at justifying a suitable host application for the plug-in; with QGIS and ArcGIS as the main options. The results from the study were that conversions between Cassini and UTM in Kenya are done using: a calculator, GIS Software, online hosted applications, and Excel Spreadsheet templates. It was determined that QGIS is the best host application since it is an open source software among other reasons.

This information was the basis for the design of a QGIS plug-in that converts between Cassini and UTM coordinate systems. The application was developed using Python Programming language. The method used involved computing 6 transformation parameters: scale, rotation, 2 translation elements and 2 other unknowns and compiling them into a csv table (acts as the plug-in database).

The developed plug-in works for Cassini and UTM coordinates extending from Longitude 36.5°, Latitude -1.0° to Longitude 36.75°, Latitude -1.25°. It is extendable to work for the whole country without any modifications to the main software code. This can be done by populating the csv table (plug-in database) to include transformation parameters for the rest of the country.

CHAPTER 1: INTRODUCTION

1.1 Background

Three coordinate systems have been in use in Kenya. These are Cassini-Soldner, East African War System, and the UTM coordinate systems (Mugnier, 2000).

The East African war systems was introduced as a military system for East Africa to consolidate coordinate systems for the British Commonwealth regions in the south, east and central Africa. It was also introduced to remove disjoints in topographical mapping and grid references through regional boundaries. All the coordinates in this system have already been converted to UTM (Mugnier, 2000).

The Cassini projection was among the main topographic mapping projections in the world until the early 20th Century. It has since been largely replaced by the Transverse Mercator Projection. The projection was first developed in 1745 by César-François Cassini de Thury (Snyder, 1987).

The projection uses a system of squares with rectangular grid coordinates; instead of showing meridians and parallels (with the exception of the central meridian). The scale along the central meridian is correct to the surveyed distance, hence approximately correcting for the ellipsoid (Craig 1882; Reignier, 1957 in Snyder, 1987). In Kenya, the origins are the intersections between the equator and the odd meridians. The odd meridians serve as the central meridian for each 2º belt which extends one degree to east and west (Mugnier, 2000).

The name Cassini-Soldner is mostly used for the form of Cassini used in Topographic mapping. This is as a result of mathematical analyses by J.G. von Soldner done in the early 19th Century which resulted into more accurate ellipsoidal formulas (Snyder, 1987). The projection is mainly used for large-scale maps of places which are largely north–south in extent (Kennedy 2000).

The UTM Projection is based on the Transverse Mercator Projection. In Europe, it is also known as Gauss Kruger Projection. It was created by Johann Heinrich Lambert (1728-1777). The Universal Transverse Mercator (UTM) projection and grid were adopted by the U.S. Army in 1947 for designating rectangular coordinates on large-scale military maps of the whole world (Snyder, 1987). The projection was introduced in Kenya in 1950 by the Directorate of Overseas Surveys (D.O.S) (Mugnier, 2000).

1.2 Problem Statement

The existence of these three coordinate systems has created the need for constant coordinate system conversion particularly between Cassini and UTM. Several efforts have been made to simplify the mathematical computations involved in the conversion of these coordinates. Such efforts range from manual computations to more automatic computations using Microsoft Excel templates and GIS software all the way to online based conversion software.

In as much as there are a variety of methods to convert coordinates between Cassini and UTM, user friendly tools to facilitate these conversions are lacking in the main GIS software used in Kenya such as QGIS and ArcGIS. As a result, the average GIS user who do not have the requisite technical background may find it difficult to merge data sets in these two systems.

GIS software such as QGIS and ArcGIS lack in-built tools for facilitating conversions between Cassini and UTM coordinates in Kenya. It is under this premise that the study is based. The study aims at finding out the methods used in facilitating the conversions; the advantages and disadvantages of these methods; proposed solutions which will be implemented as a software plug-in; the best host application for the plug-in among the three software.

1.3 Objectives

The main objective of this study is to build a helper application to overcome the challenges of Cassini to UTM coordinate system conversions in Kenya

The specific objectives of the study are:

1. To establish and review the methods used by the Kenyan GIS Community to convert coordinates between Cassini and UTM coordinate systems and their associated challenges.
2. To propose solutions to the challenges associated with the methods currently used in Kenya to convert coordinates from Cassini to UTM.
3. To design a QGIS helper application to implement solutions to the observed challenges in converting coordinates from Cassini to UTM and vice-versa.

1.4 Justification for the Study

The existence of multiple coordinate systems in Kenya has caused the need for constant coordinate system conversion mainly between Cassini and UTM coordinate systems. This has led to a variety of methods to facilitate these conversions. These methods are either complex due to the tedious computations involved or slow since only a few coordinates can be transformed at a time. As a result, they are difficult to use by the average GIS user.

This research has the main purpose of carrying out a study that will lead to the development of a helper application that greatly simplifies this process of coordinate system conversion between Cassini and UTM in Kenya. This application will be hosted in GIS software and should have the advantage of being user friendly; easy to learn and use. It should provide great benefits for average GIS users who do not have the technical knowhow of carrying out the complex mathematics of transforming coordinates between UTM and Cassini. This will make it easy for GIS users to easily and quickly merge datasets within these two coordinate systems.

1.5 Scope of work

The study involved carrying out a desk study which includes a review of the existing literature. This had the main aim of finding out how far the research community has gone in providing solutions to the challenge of Cassini and UTM coordinate system conversion in Kenya. The information obtained from the study will led to the identification gaps not covered by the current solutions. These gaps formed the basis for the design of the light application that will extend the host GIS software functionality by enabling the conversion of coordinates between Cassini and UTM in Kenya.

CHAPTER 2: LITERATURE REVIEW

2.1. Cassini Projection

The Cassini Projection is Cylindrical and neither equal area nor a conformal projection. This projection has the Central Meridian and the equator as straight lines; other meridians and parallels are complex curves. Each of the meridians are 90° from the central meridian. The scale is true along the central meridian, and along lines perpendicular to the central meridian. On spherical form of the projection, the scale is constant but not true along lines parallel to the central meridian. This is nearly so for the ellipsoidal form (Snyder, 1987).

The scale on the Cassini Projection steadily increases in a course parallel to the central meridian, as the distance from that meridian increases, but it is constant along any straight line on the map which is parallel to the central meridian. Because of this, the Cassini Projection is appropriate for areas that are mostly north-south in extent, such as Great Britain, than for areas extending in other directions. Thus, it is similar to the Transverse Mercator (Snyder, 1987).

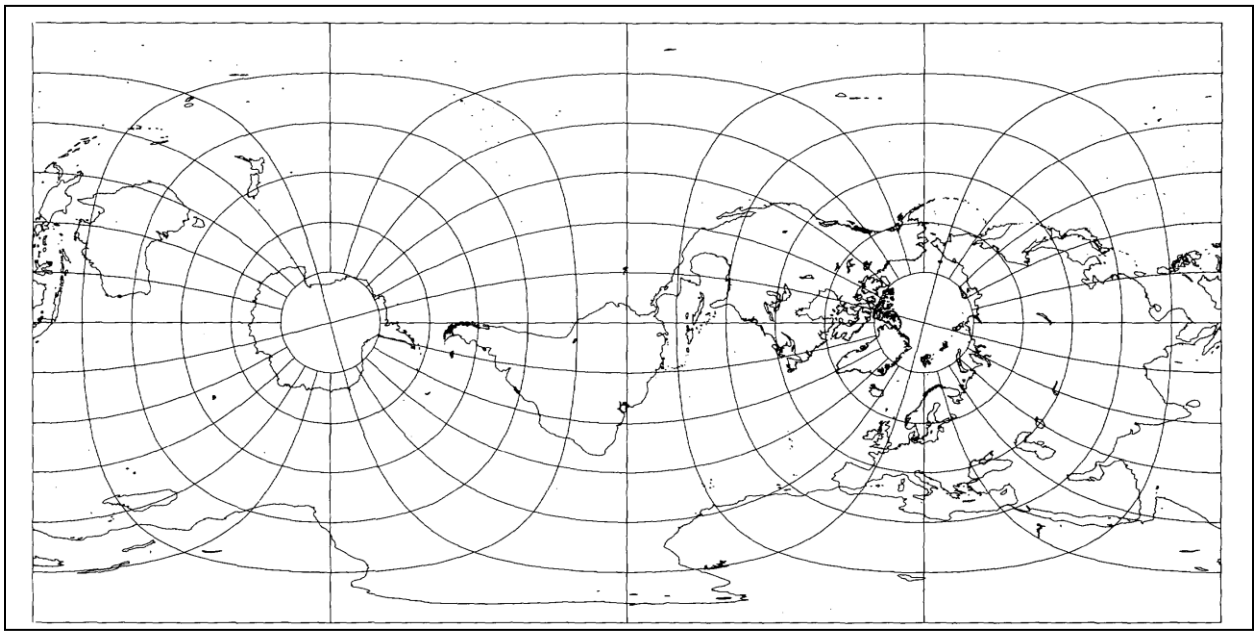


Figure 1: Cassini Projection, Source: (Snyder, 1987)

Cassini Projection in Kenya

The Cassini Soldner coordinate system was made known in Kenya during the colonial times. Consequently, nearly all triangulation networks earlier than 1950 were established on this system. As a result, the cadastral surveys in Kenya done in this system (Kamau, 2016). From about the year 1920, the Ordnance Survey started using the Transverse Mercator due to the difficulty in measuring scale and direction on Cassini (Snyder, 1987).

For the Cassini projection in Kenya, the origins are the intersections between the equator and the odd meridians. The odd meridians serve as the central meridian for each 2º belt which extends one degree to east and west. The reference ellipsoid used is Clarke 1858 while the unit of measurement is the British foot (Mugnier, 2000). Table 1 gives a summary of Cassini Projection Parameters in Kenya.

Table 1: Cassini Projection in Kenya (Gacoki, 2013)

No.	**Parameters**	
1	Flattening, *f*	0.003
2	eccentricity, e^2	0.007
3	Semi major axis, *a*	20926348.000
4	Semi minor axis, *b*	20855232.837
5	Spheroid	Clarke 1858
6	Projection	
7	Unit of Measure	Feet
8	Meridian of Origin	37° E
9	Latitude of Origin	Equator
10	Datum	Modified (1960) arc

2.2. Universal Transverse Mercator projection

The Universal Transverse Mercator Projection (UTM) is founded on the Mercator projection. The projection is conformal. It is defined from 84°N and 80°S and is distributed into 60 Zones, each with a 6° wide longitude. Continuing from the 180th Meridian, the zones are given numbers 1 to 60; with minor exceptions. The latitudinal zones are 8 degrees wide, except for zone X which is 12 degrees wide. The latitudinal zones are designated with the letters C to X (skipping I and O). As a result, quadrangles are formed. These quadrangles are subdivided further into 100,000 meter wide grid squares on a side with a double letter designation (Snyder, 1987).

Geographic locations in UTM are given x and y coordinates in meters. The meridian halfway between two bounding meridians is used as the central meridian, and the scale is reduced to 0.9996 of the true scale to minimize scale deviation in a given zone. When defining location of a point, the zone number, x and y coordinates are adequate. The 100,000m square description is not needed if the ellipsoid and the hemisphere (North or South) are known (Snyder, 1987).

The UTM Coordinate System is created on a cylinder that is rotated making the tangent line a meridian of longitude. The scale error is zero on the standard lines and 0.0004 (1 - 0.9996) on the central meridian. The scale grows as you move east or west of the standard lines. The maximum scale error is controlled by only using the projection +/- 3 degrees of longitude of the central meridian (Snyder, 1987).

In the Northern Hemisphere, the Equator at the Central Meridian is taken to be the origin. It has an assigned an x coordinate of 500,000m and a y coordinate of 0 m. In the Southern Hemisphere, x is assigned 500,000m and y 10,000,000m. Thus, negative coordinates are avoided in both the Northern and Southern Hemispheres because the numbers increase East and North. The UTM Projection uses the ellipsoidal earth. The reference ellipsoids change with a specific region of the earth (Snyder, 1987).

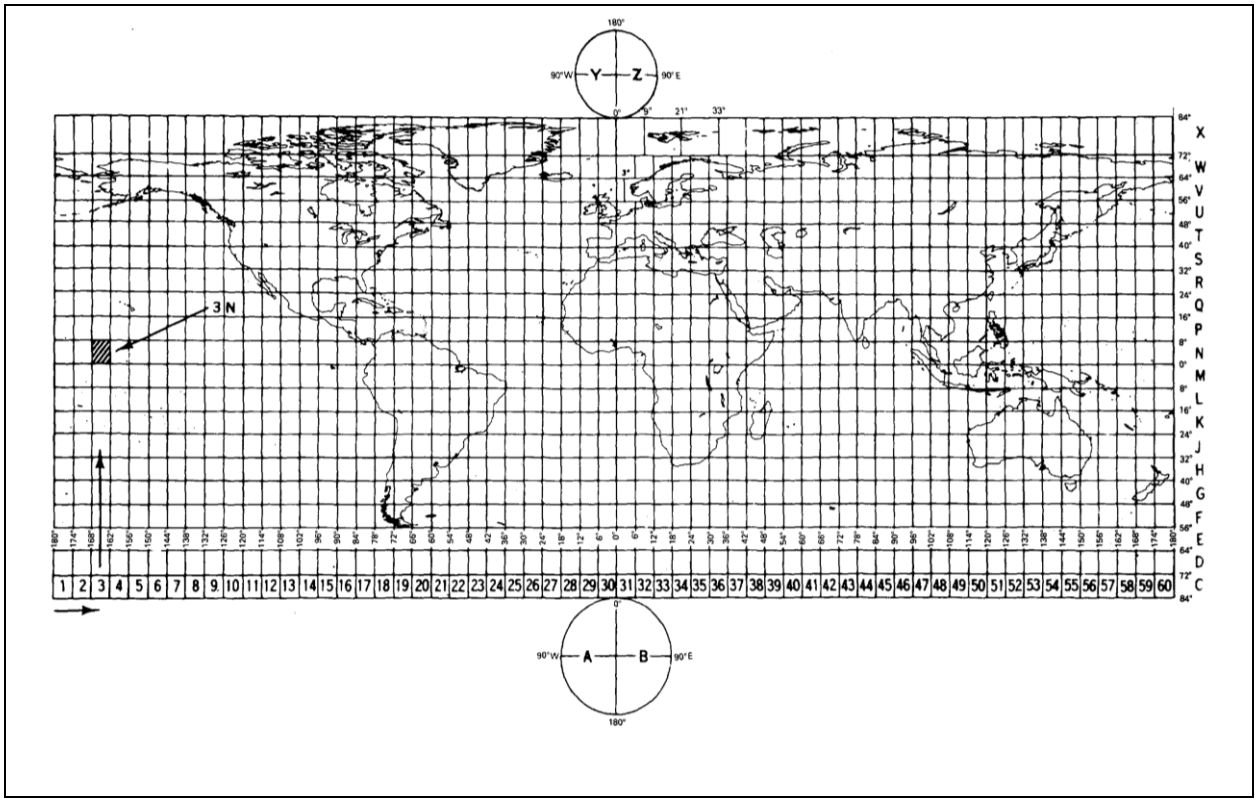


Figure 2: UTM Projection, Source: (Snyder, 1987)

UTM Projection in Kenya

The UTM projection was introduced in Kenya by the Directorate of Overseas survey (D.O.S) in 1950. This was when the D.O.S begun offering survey services in Kenya. The system uses Clarke 1880 spheroid; whereas the unit of measurement is the international meter. Several efforts have been made by the Survey of Kenya to convert all points to this coordinate system. Table 2 gives a summary of UTM Coordinate system parameters in Kenya.

Table 2: UTM Coordinate System in Kenya (Gacoki, 2013)

No.	**Parameters**	
1	Flattening, *f*	0.003407561
2	eccentricity, e^2	0.006803511
3	Semi major axis, *a*	6378249.145 m
4	Semi minor axis, *b*	6356514.87 m
5	Spheroid	Clarke 1880
6	Projection	Transverse Mercator

7	Unit of Measure	Meter
8	Meridian of Origin	39° E
	Latitude of Origin	Equator
9	Scale factor of Origin	0.9996
10	False Origin	500000W
		10000000S
11	Datum	Modified (1960) arc

2.3. Coordinate Transformation Formulas

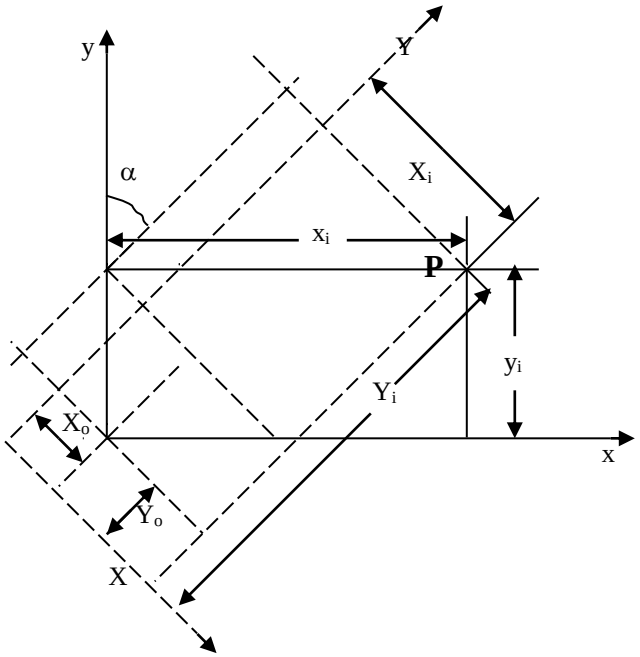


Figure 3: Coordinate Transformation, Source: (Gacoki, 2013)

Consider a point P with coordinates (X_i,Y_i) on one coordinate system and (x_i,y_i) in the other as shown in figure 1, the coordinates (X_i,Y_i) in terms of (x_i,y_i) are given by

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = k\begin{pmatrix} x_i\cos\alpha & -y_i\sin\alpha \\ x_i\sin\alpha & y_i\cos\alpha \end{pmatrix} + \begin{pmatrix} X_o \\ Y_o \end{pmatrix} \qquad (2.1)$$

This can be written as:

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} kcx_i & -ksy_i \\ ksx_i & kcy_i \end{pmatrix} + \begin{pmatrix} X_o \\ Y_o \end{pmatrix} \quad (2.2)$$

Where,

k is the scale factor

c is the cosine of the rotation angle α

s is the Sin of the rotation angle α

X_o, Y_o are the translation elements

These 4 parameters (k,α, X_o, Y_o) have to be determined so as to perform a coordinate transformation. Nevertheless, in some cases, it is not enough to assume a linear relationship between 2 sets of coordinates especially if they are on different types of projections. In such a case a second order transformation is necessary. The general second degree polynomial is given by: -

$$ax^2 + by^2 + 2hxy + gx + fy + c \quad (2.3)$$

This general second order degree polynomial is added to equation (2.2), to obtain

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} Px_i - Qy_i + Rx_i^2 + Sy_i^2 + 2Dx_i y_i \\ Qx_i + Py_i + Ex_i^2 + Fy_i^2 + 2Gx_i y_i \end{pmatrix} + \begin{pmatrix} C_x \\ C_y \end{pmatrix} + \begin{pmatrix} v_{xi} \\ v_{yi} \end{pmatrix} \quad (2.4)$$

Where,

P=kc and Q=ks, Cx =Xo and Cy =Yo. Equation (2.4) can be written in matrix form as follows.

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} x_i & -y_i & x_i^2 & y_i^2 & 2x_i y_i & 0 & 0 & 0 & 1 & 0 \\ y_i & x_i & 0 & 0 & 0 & x_i^2 & y_i^2 & 2x_i y_i & 0 & 1 \end{pmatrix} \begin{pmatrix} P \\ Q \\ R \\ S \\ D \\ E \\ F \\ G \\ Cx \\ Cy \end{pmatrix} + \begin{pmatrix} v_{xi} \\ v_{yi} \end{pmatrix} \tag{2.5}$$

From equation (5), 10 unknowns need to be computed i.e. P, Q, R, S, D, E, F, G, Cx, Cy to solve the problem with a minimum of 5 common points.

Since $R \approx -S \approx G$ say $R = A$ and $D \approx -E \approx F$ say $D = -B$; this reduces the parameters to be determined from 10 to 6 i.e. P, Q, A, B, C_x and C_y. Equation (2.4) can therefore be written follows:

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} Px_i - Qy_i + A(x_i^2 - y_i^2) - 2Bx_i y_i \\ Qx_i + Py_i + 2Ax_i y_i + B(x_i^2 - y_i^2) \end{pmatrix} + \begin{pmatrix} Cx \\ Cy \end{pmatrix} + \begin{pmatrix} v_{xi} \\ v_{yi} \end{pmatrix} \tag{2.6}$$

Finally, equation (2.6) can be written in matrix form as: -

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix} = \begin{pmatrix} x_i & -y_i & x_i^2 - y_i^2 & -2x_i y_i & 1 & 0 \\ y_i & x_i & 2x_i y_i & x_i^2 - y_i^2 & 0 & 1 \end{pmatrix} \begin{pmatrix} P \\ Q \\ A \\ B \\ C_x \\ C_y \end{pmatrix} + \begin{pmatrix} v_{xi} \\ v_{yi} \end{pmatrix} \tag{2.7}$$

A minimum of 6 equations is essential to determine the 6 unknown parameters. For a least squares solution, a minimum of 3 points with coordinates in one system and a corresponding number in the other system is required. Once the equations are formed, they are normalized in the usual way, and a solution to the 6 parameters obtained (Gacoki, 2013).

Converting non-conformal Cassini to conformal Cassini coordinates

To convert non-conformal Cassini coordinates to conformal Cassini coordinates a correction is added to the Cassini eastings as follows:

$$x_c = \frac{x^3}{6R^2} + \frac{x^5}{24R^4} + \ldots \tag{2.8}$$

Where R is the radius of the earth and is given by

$$R = \sqrt{a * b} \tag{2.9}$$

a, b are the semi-major and semi-minor axis of the ellipsoid respectively (Gacoki, 2013).

2.4. Similar Research

This study is similar to a research presented in two papers by (Gachoki T. G., 2013). The first paper outlines a method of converting Cassini to UTM coordinates using the excel spreadsheet. The second paper outlines the same method in converting UTM to Cassini coordinates. The main difference between this study and these papers is that he uses the excel spreadsheet to facilitate the conversions whereas this study uses a QGIS helper application to facilitate the conversions.

The first paper by (Gachoki T. G., 2013) outlines the steps for converting Cassini grid coordinates (planimetric only) to UTM coordinates on the Excel spreadsheet. A general second degree polynomial is used to calculate 6 parameters which include a scale, rotation 2 translation elements and 2 other unknowns. Three interconnected worksheets are used to calculate the transformation. The first excel worksheet is used for entering data. The second worksheet is used to calculate the 6 transformation parameters necessary for the transformation by use of entries in the first worksheet. The third worksheet is used to calculate the transformed coordinates in UTM by use of the data entered in the first worksheet and the calculated parameters from the second worksheet.

The second paper by (Gachoki T., 2013) presents the method for converting UTM coordinates to Cassini grid coordinates on the excel spreadsheet. The conversion of UTM coordinates to Cassini coordinates follows the same procedure as that one of converting Cassini to UTM with minor differences.

CHAPTER 3: MATERIALS AND METHODS

3.1 Study Area

The area of study extends from geographical coordinates 36.5°,-1.0° to 36.75°,-1.25°. It is covered by topographic map sheet 148/1 (Limuru), with a total of 769.89 Km^2; see figure 4.

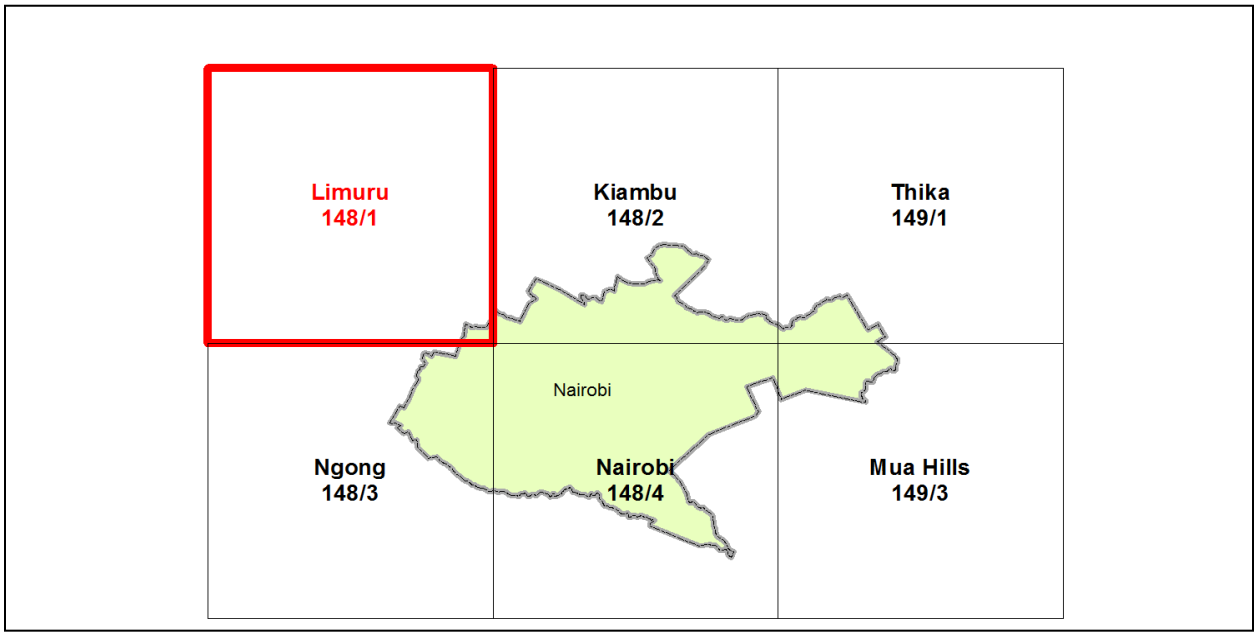


Figure 4: Location of study area, Source: Survey of Kenya

3.2 Methodology

The methodology used in the study was split into three stages in line with the objectives. These are the Desk Study, the Findings and the Proposals. The schematic presentation of the methodology is presented in figure 5.

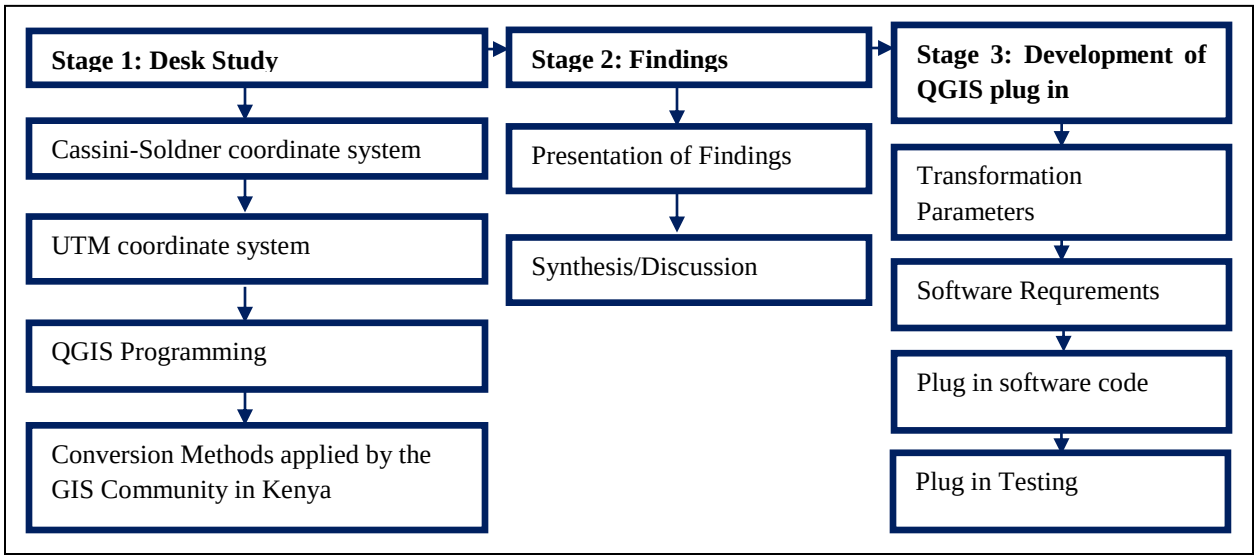


Figure 5: Methodology

3.2.1. Stage 1: Desk Study

Stage 1 of the methodology was a desk study designed to address the first objective: *'To establish and review the methods used by the Kenyan GIS Community to convert coordinates between Cassini and UTM coordinate systems and their associated challenges'.* The study involved carrying out a review of the existing literature with the main aim of finding out how far the research community has gone in providing solutions to the challenge of Cassini and UTM coordinate system conversion in Kenya.

It involved a review of the Cassini Soldner and the UTM coordinate system. The review was conducted on information from the Survey of Kenya; Online Sources such as research papers, books and websites; and books from the University of Nairobi Library.

The study also involved a review of information on GIS software available in Kenya such as QGIS and ArcGIS. This was done with the main purpose of finding out how to program and customize the software, the possible programming languages that can be used, and the best application to host the plug in among the available GIS software. This information was sourced from research papers, books, main application websites and video tutorials.

The conversion methods applied by the Kenyan GIS community were reviewed from online sources to get a clear picture of how far the society has gone in providing solutions to converting

coordinates between Cassini and UTM. The objective was to get the current methods being used, their challenges and their possible solutions or improvements.

3.2.2. Stage 2: Findings

Stage 2 of the methodology was designed to address the first objective: *'To establish and review the methods used by the Kenyan GIS Community to convert coordinates between Cassini and UTM coordinate systems and their associated challenges'*. This was done by analyzing and presenting the findings from stage 1 so as to gain new insights and to facilitate the making of proposals.

In this stage, the information collected and reviewed from existing sources was analyzed compiled and presented in the form of text, tables and mathematical expressions. The analysis involved a comparison of the major GIS software available in Kenya, such as QGIS and ArcGIS for the purpose of finding out the best software to host the plug in. It also involved a comparison of the methods used by the Kenyan GIS community in converting coordinates between Cassini and UTM in order to get the advantages and disadvantages and optimal solutions. Conclusions were drawn from the analyzed data and new insights were gained. These new insights helped formulate proposals which were implemented as a plug in.

3.2.3. Stage 3: Development of QGIS plug in

Stage 3 of the methodology was designed to address the third objective: *'To design a QGIS helper application to implement solutions to the observed challenges in converting coordinates between Cassini and UTM'.*

In this stage of the methodology, the new insights obtained from the information in stage two was used to formulate proposals as solutions to the identified challenges. These were implemented as Plug in which was hosted in the QGIS software as was determined. The language chosen to program the software was Python. This mainly is because Python is the primary scripting language for QGIS.

The QGIS plug in development process involved: calculating the transformation parameters, downloading and installing relevant programming software, developing the plug in software code, and plug in installation and testing; See figure 5.

Calculating the Transformation Parameters

The transformation parameters (a, b, Tx and Ty) were calculated using the formulas 2.1 to 2.9 and 3.10 to 3.13. A minimum of four Cassini and four UTM Sheet Corners were used to calculate transformation parameters for each of the 25 Cassini and UTM grids present on Sheet 148/1. These parameters were then compiled into two csv tables; one for Cassini to UTM and the other for UTM to Cassini. These csv tables act as a database holding the transformation parameters used in the plug-in.

Software Requirements

Once the transformation parameters were calculated and compiled into tables, the software and system requirements needed to develop the plug in were determined, downloaded and installed. The relevant software include the following; QGIS, Python, Qt, PyQT, PyQt Development tools, Text editor (Python IDE), Plug in builder plug in for QGIS, Plug in re-loader plug in for QGIS.

Developing the Plug in Software Code

Qt Creator was used to design the user interface for the QGIS plug in. The Plug-in Builder was installed into QGIS and used to create all the essential files and the boilerplate code for a plug-in. The Plug-in Re-loader plug-in was also installed into QGIS and used during the main software code development to allow iterative development of plug-in. This allowed changes in the plug in code to be made and reflected in QGIS immediately. This eliminated the need to restart QGIS every time there was a code change. Python software was installed and the Python IDE was used to edit the software code for the plug in.

Plug in Testing

The iterative development of the plug in allowed each line of the code to be tested as development was in progress. Once the code was complete and confirmed to be working on the computer system used in programming the plug in, it was installed on another other computers with different specifications to ensure it would work on various computer systems.

CHAPTER 4: RESULTS AND DISCUSSIONS

4.1. GIS software

The major commercial off the shelf GIS software are provided by ESRI (ArcGIS); Intergraph (GeoMedia); Bentley Systems; Smallworld Systems; Autodesk (AutoCAD Map); Graphic Data Systems (GDS); ERDAS/Imagine, ER MAPPER, PCI, Envi, Genassys II; GRASS;

Global mapper, developed by Blue Marble Geo-graphics, has a minor but a notable market share. In addition, there are over 38 notable GIS software that fall within the category of open source GIS. The major open source GIS software used in Kenya is QGIS. Figure 6 shows the GIS software market.

GIS Market Share (%)

VENDOR/YEAR	1992	1993	1994*	1995	1996**
ESRI			30.3		32.5
Intergraph			24.1		26.5
MapInfo			5.6		5.1
GDS			4.9		6.5
Atlas (Strat. Map)			4.9		
IBM			3.3		
Enghouse			2.2		
ERDAS			2.2		2.7
Genasys			2.1		
PCI			2		
Other			18.4		
Total ($millions)			495	548	591

Figure 6: GIS Software Market, source; (**Briggs, 1998)**

4.1.1. ArcGIS

ArcGIS is an ESRI software product that has a scalable and comprehensive structure for executing GIS for one or many users on the desktop, server, over the web or in the field. It is a

combined collection of GIS software products consisting of ArcGIS Desktop, ArcGIS Engine, Server GIS, and Mobile GIS frameworks (ESRI, 2004).

ArcGIS Desktop

ArcGIS Desktop is a unified suite of advanced GIS software which include a series of Windows Desktop software such as ArcMap, ArcCatalog, ArcToolbox, ArcScene, and ArcGlobe; all of which have user interface components. These interfaces can be used to carry out any GIS task, simple or advanced, such as mapping, geographic analysis, data editing and compilation, data management, visualization, and geo-processing (ESRI, 2004).

ArcGIS Desktop is offered at three practical levels: ArcView, for comprehensive data use, mapping and analysis; ArcEditor, advanced geographic editing and data creation; and ArcInfo, a complete professional GIS desktop, containing wide-ranging GIS functionality, including rich geo-processing tools (ESRI, 2004).

ArcMap

ArcMap is the main application in the ArcGIS Desktop for all map based tasks such as cartography, map analysis, and editing. It is a comprehensive map authoring software for ArcGIS Desktop (ESRI, 2004).

4.1.2. QGIS

QGIS is a free and open source GIS which supports creating, editing, visualizing, analyzing and publishing of geospatial information on Windows, Mac, Linux, and BSD Operating Systems for desktop, server, web browser and developer libraries (QGIS, 2017). QGIS is maintained and developed by volunteers who regularly release updates and fix bugs.

The software supports multiple file formats such as shape-files, coverages, personal geodatabases, dxf, MapInfo, and PostGIS. It also supports web services including Web Map Service and Web Feature Service to allow use of data from external sources. The software integrates with other open-source GIS packages, including PostGIS, GRASS GIS, and MapServer.

4.1.3. Choice of Software

ArcGIS and QGIS offer the best software platforms to host the plug in. ArcGIS is a commercial off the shelf software. It's expandable and has a wide user community to find answers. The software offers tutorials with sample data for a user to get practical experience. Unlike QGIS, the license level determines which tools a GIS user can use.

QGIS is an open source software; does not have license levels. As a result, the software does not limit which tools can be used. It has multi-language support and relies on volunteer efforts for development. Though QGIS is not as advanced as ArcGIS in terms of solutions to GIS problems, it offers the best chance for GIS users to easily access any new tool. Thus it is the best software to host the plug in. QGIS offers the following advantages:

1. QGIS is Open Source; no license levels in QGIS hence does not limit which tools can be used.
2. QGIS uses the GDAL/OGR library to read and write GIS data formats (Over 70 vector formats supported)
3. QGIS has a QGIS Browser as a stand-alone GIS data management application
4. QGIS has support for 2,700 known coordinate reference systems (CRS). It allows a user to define global and project-wide CRS for layers without a pre-defined CRS. It also allows a user to define custom CRS and supports on-the-fly projection of vector and raster layers.
5. QGIS has over 300 plug-ins to engineer specialized analyses. The software has plug-in re-loader and plug in builder plug-ins making it very easy for GIS users to create new plug-ins. It also has a plug in manager for installing and managing plugins (GIS Geography, 2017).

4.2. GIS Programming

GIS programming forms a vital element in the design and use of GIS software systems. It is required when interacting with GIS database systems, converting between data formats, changing between projections, solving problems (data analysis), and automating common repetitive tasks.

4.2.1. GIS Programming languages

There is a surprising variety of programming languages which are useful for geospatial professionals today; over 600 languages (The Pennsylvania State University, 2017), excluding the byzantine diversity of dialects of BASIC past and present. The most relevant to GIS include Python, Java Script, C++, Java, C, SQL, C#, Visual Basic.NET, Flex, ActionScript, PHP, R and S, Ruby, Groovy, Jython, Scala, Avenue, and VBA for ArcObjects.

GIS programming languages can be categorized into three: Object Oriented, Procedural, and Scripting languages. Java is a fully Object Oriented programming language, whereas Python and C++ support Object Oriented programming; C is Procedural a programming language; and Python, Perl, Ruby, and JavaScript are Scripting languages.

Python

Python is a dynamic interpreted language with direct execution; no compilation. The language is often used as a scripting language in automating and simplifying tasks, as well as building large complex programs. It is presently gaining popularity as the primary scripting language for ArcGIS and QGIS (The Pennsylvania State University, 2017).

SQL

SQL is a database access and control language hence the heart of many GIS operations. It is a declarative language; with statements tell what you want to happen, as opposed to how you want it to happen (The Pennsylvania State University, 2017).

Java Script

Java Script is mainly used in developing Web User Interfaces such as Google Maps and Open Layers. The language can also be used as a Scripting language in interpreting and automating task execution. It can also be used in a fully Object Oriented way (The Pennsylvania State University, 2017).

C++

C++ is a systems programming language, supporting both procedural and object oriented programming. A vast majority of software in use every day are written in C++ such as ArcGIS,

GRASS GIS, QGIS, Windows OS, Firefox, MS Office, etc (The Pennsylvania State University, 2017).

Java

Java is a popular language for web programming, and is the language of choice for most programmers. It is built on C++ but significantly simplified and only supports object oriented programming. It is one of the contenders for the most popular Open Source GIS languages such as GeoServer and JTS projects. Java is also one of the most commonly taught programming language in universities (The Pennsylvania State University, 2017).

Choice of Programming Language

Both QGIS and ArcGIS offer Python support making it the primary scripting language for data analysis, data conversion, data management, and map automation. Thus it becomes the best choice of language for developing the Cassini and UTM coordinate system conversion plug-in. Python offers the following advantages:

- It is easy to learn and use
- Highly scalable, suitable for large projects or small one-off programs (scripts)
- It is portable, cross-platform
- It is embeddable (making QGIS scriptable)
- It is stable and mature
- It has a large user community (ESRI, 2017)

4.2.2. GIS Programming in ArcGIS

ArcGIS allows users to create their own customized extensions by working with ArcObjects, the ArcGIS software components library. The extensions and custom tools are developed using standard Windows programming interfaces such as Visual Basic (VB), .NET, Java, and C++ (ESRI, 2004). In ArcGIS, Python scripting is used for automating tasks (through running Python scripts), as well as writing applications, such as add-ins. Hence it is possible to write scrips and plug ins to carry out coordinate system transformations which are not supported by the software.

4.2.3. GIS Programming in QGIS

QGIS has an optional scripting interface using the Python Programming language, the Integrated Python Console. This interface allows users to create and use scripts for automating processes instead of doing similar tasks repeatedly (QGIS Project, 2014).

QGIS also allows for the extension of its functionality by the use of plug-ins. Initially only possible using C++ programming language. C++ has the advantage of simplicity of distribution (no compiling for each platform needed) and easier development (QGIS Project, 2014).

The addition of Python Programming Language Support made it possible for users to write and use plug-ins written in Python. Using Python Programming Language has the advantage of being easy to write, understand, maintain and distribute due the dynamic nature of the language (QGIS Project, 2014).

QGIS has a plug-in installer that allows users to fetch, upgrade and remove plug-ins (QGIS Project, 2014).

The python support available in QGIS enables the writing of scripts and development of plug-ins to facilitate coordinate system conversions; for those not supported by the software.

4.3. Current Conversion Methods

There are several methods used by the Kenyan GIS Community to convert coordinates between Cassini and UTM. These include:

1. Manual Computations using a calculator
2. GIS Software
3. Online tools
4. Using the Excel Spreadsheet

4.3.1. Manual Computations

It is possible to manually convert between UTM and Cassini Coordinate Systems by carrying out computations manually using a scientific calculator. This method is is highly prone to human error; it is tedious and time consuming since only one coordinate can be converted at a time; and is limited to GIS users who are conversant with the conversion formulae.

4.3.2. GIS Software

Many of the Commercial off the self as well as open source GIS software have tools that allow for the conversion of coordinates from one projection to another. Most of the major projections used globally are supported within these softwares, with the exception of projections specific to a certain country.

In this case, there is an option on some of the softwares to add the respective transformation parameters to facilitate the conversion. This mehod is much better than the manual one but has the main disadvantage of having to key in transfromation parameters everytime one has to do a transformation. As a result, it is also prone to human error.

The QGIS software has several plug ins that support the coonversion of coordinates between projections not supported within the main software. An examples includes the Surveying Calculation plugin for QGIS 2.x.

The Surveying Calculation plugin for QGIS 2.x was developed for the Land Parcel Cadaster of Zanzibar. It provides coordinate transformation based on common points having coordinates in both coordinate systems. Two separate coordinate lists have to be created with the coordinates in the two coordinate systems before starting the coordinate transformation (DigiKom Ltd, 2014).

This method has the advantage of being free of human error. The main disadvantage is that a user needs to have a set of coordinates in both systems to facilitate the transformation. This means incase a user has only one set of coordinates from one system, then they will not be able to do the conversion.

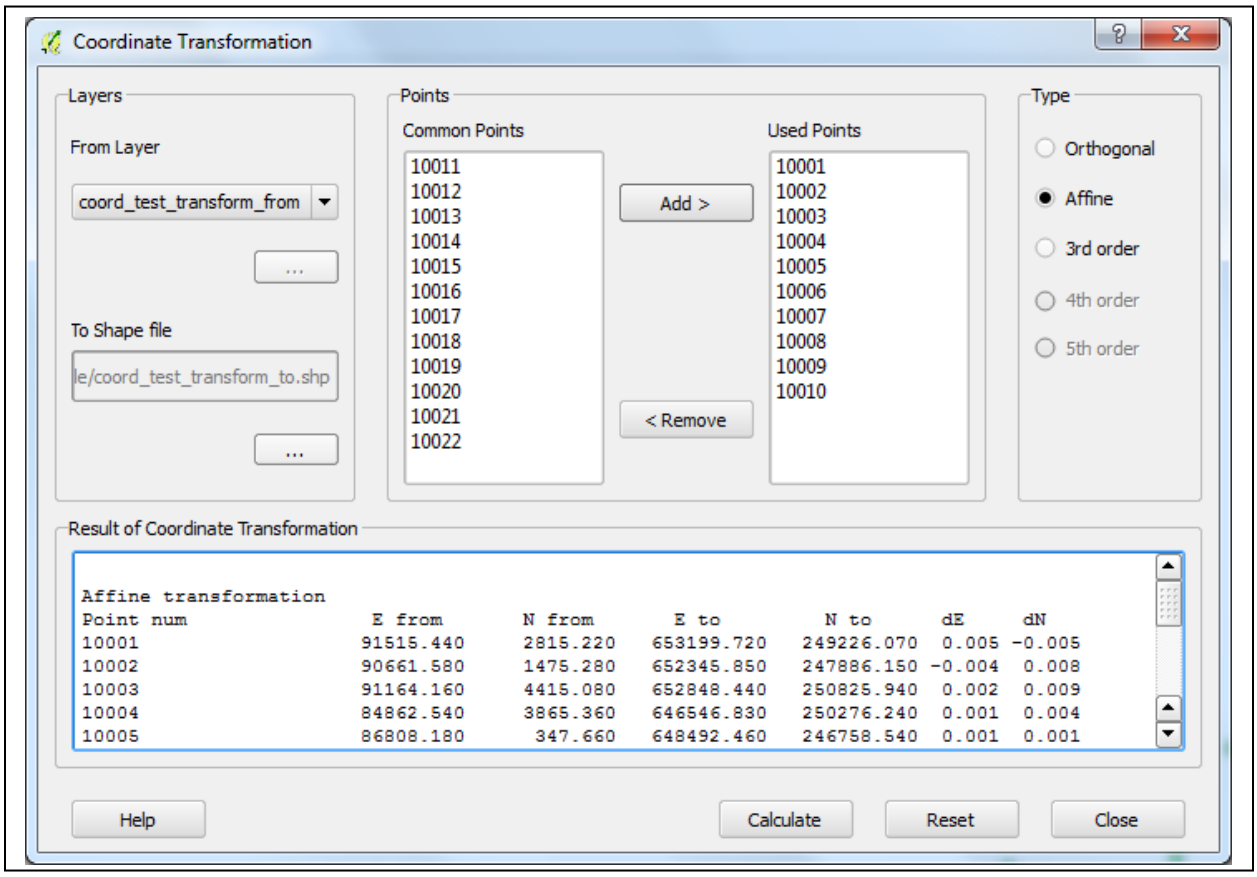


Figure 7: SurveyingCalculation plugin for QGIS 2.x; Source: (DigiKom Ltd, 2014)

4.3.3. Online tools

There are various online hosted applications that facilitate the transformation between different Projections. These work for most parts of the earth and where a perticular location is not facilitated, one also has the option of adding the transformation parameters for that region or country. It is much better since the parameters will be kept in a database for future use. Thus, once entred, the user doese‘nt have to enter them again.

The main disadvantage of this method is that one has to enter the transformation parameters incase the coordinates fall in an area not covered by the application. The user also needs internet access to use this aplicaion.

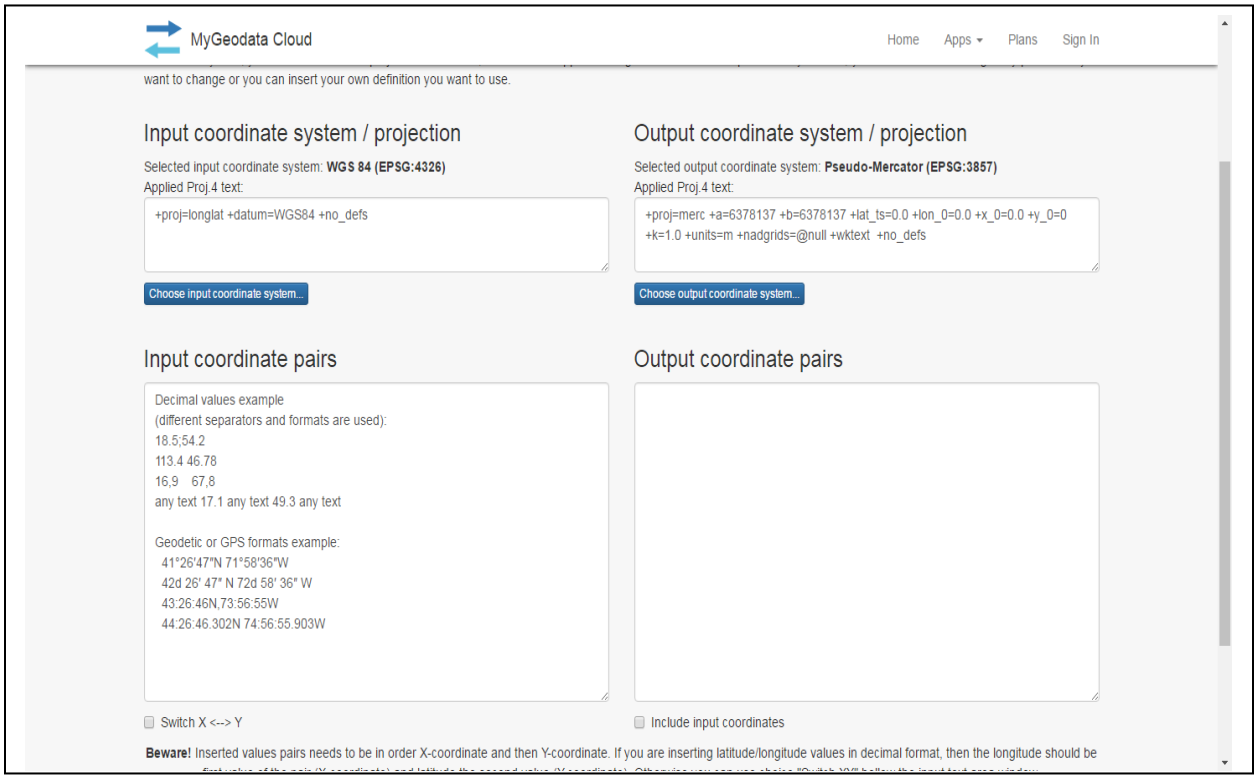
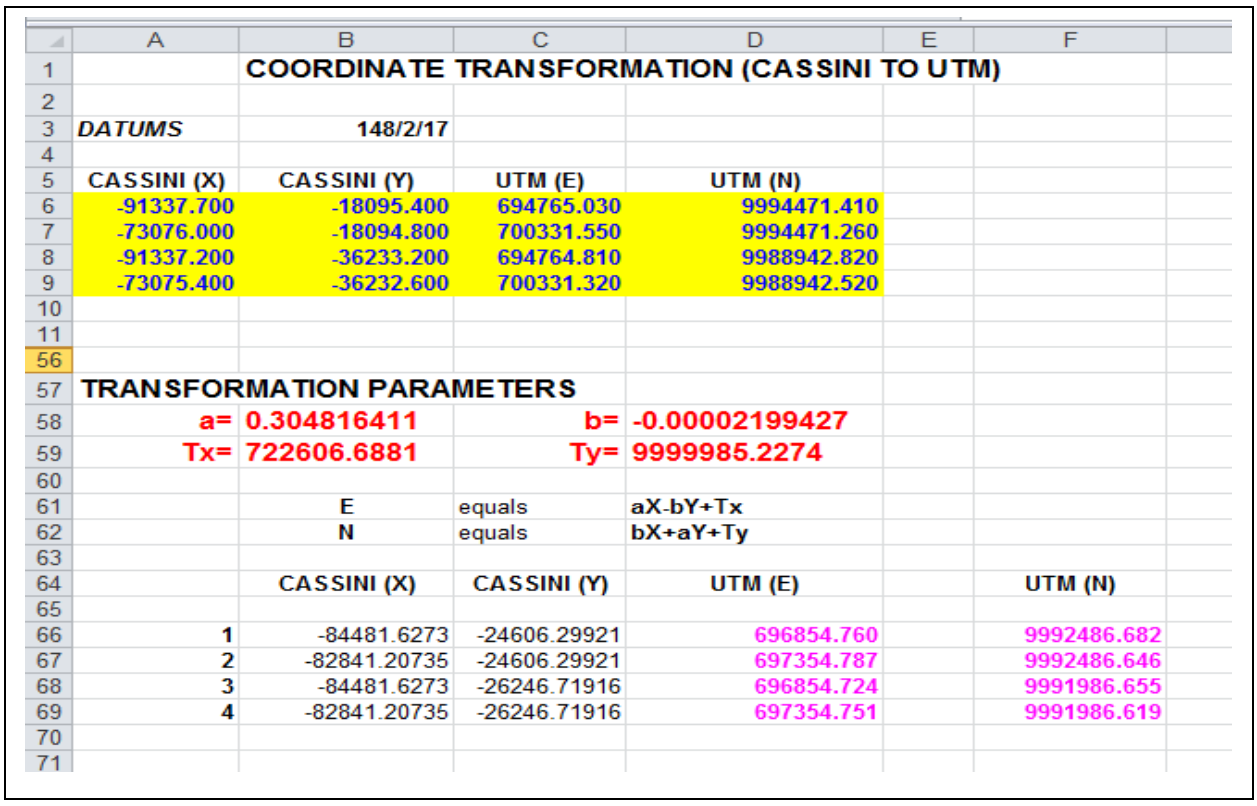


Figure 8: An online based Coordinate Transformation Application; Source: (MyGeodata Cloud, 2016)

4.3.4. The Excel Spreadsheet

The excel spread sheet makes it easier to convert between Cassini and UTM projections by use of a template already created by Gacoki, 2013; Outlined in the report *'Conversion of Cassini Coordinates to UTM Coordinates on the Excel Spreadsheet.'*. The method is easier to use than the manual computation and more than one coordinate can be converted at a time.

It has the disadvantage of having to type in specific sheet corners that apply to the coordinates being converted so as to generate transformation parameters. As a result, it is also prone to human error especially if one enters the wrong sheet corners. Also, a user will not be able to do any conversions if they do not have the necessary sheet corners.



	A	B	C	D	E	F
1		COORDINATE TRANSFORMATION (CASSINI TO UTM)				
2						
3	*DATUMS*	148/2/17				
4						
5	CASSINI (X)	CASSINI (Y)	UTM (E)	UTM (N)		
6	-91337.700	-18095.400	694765.030	9994471.410		
7	-73076.000	-18094.800	700331.550	9994471.260		
8	-91337.200	-36233.200	694764.810	9988942.820		
9	-73075.400	-36232.600	700331.320	9988942.520		
10						
11						
56						
57	TRANSFORMATION PARAMETERS					
58	a=	0.304816411	b=	-0.00002199427		
59	Tx=	722606.6881	Ty=	9999985.2274		
60						
61		E	equals	aX-bY+Tx		
62		N	equals	bX+aY+Ty		
63						
64		CASSINI (X)	CASSINI (Y)	UTM (E)		UTM (N)
65						
66	1	-84481.6273	-24606.29921	696854.760		9992486.682
67	2	-82841.20735	-24606.29921	697354.787		9992486.646
68	3	-84481.6273	-26246.71916	696854.724		9991986.655
69	4	-82841.20735	-26246.71916	697354.751		9991986.619
70						
71						

Figure 9: Coordinate Transformation on the Excel Spreadsheet

4.3.5. Observed Challenges

Cumulatively, three main challenges have been observed with the current conversion methods. These are:

1. The methods are prone to human error as a result of having to manually type in coordinates, transformation parameter or compute the conversions manually;

2. The manual computations using a calculator are tedious and time consuming due to the technical computations involved; making it difficult for users who are unfarmiliar with the computations;

3. Online hosted tools require internet connection hence cannot be accessed by users in areas with low or no internet connectivity.

4.4. Proposed solutions

The following are the proposed solutions to the observed challenges:

The solution should be an application that is hosted on a GIS software system that lacks an in-built mechanism of converting between Cassini and UTM Coodinate Systems. This will solve the problem of having to compute the conversions manually.

The proposed method should make it easy to convert between the Cassini and UTM Coordinate Systems without having to add in any sheet corners or transformation parameters. These should be contained in a database within the software. The application should also be able to work with any of the major file formats such as csv, or shapefile. This solves the human error problem that comes about as a result of keying in coordinates or transformation parameters.

In addition, the application should be freely available to the public and usable even without internet conection. This solves the accessibility problem as a result of poor internet connection or having to buy the software.

4.5. QGIS Helper application Design

4.5.1. Transformation Parameters

The transformation parameters (a, b, Tx and Ty) were calculated using the formulaes 2.1 to 2.9 and 3.1 to 3.2. A minimum of four Cassini and four UTM Sheet Corners were used to calculate the parameters for each of the 25 Cassini and UTM grids present on Sheet 148/1. These were then compiled into two csv tables one for Cassini to UTM and the other for UTM to Cassini. These csv tables act as a database holding the transformation parameters used in the plug-in. See Tables 3 and 4.

Table 3: Cassini to UTM Transformation Parameters for Limuru Sheet 148/1

CASSINI TO UTM TRANSFORMATION PARAMETERS				
Grid	**a**	**b**	**Tx**	**Ty**
1	0.3049639225937430	0.00019116102703264900	277458.9711691740	10000238.10112570
2	0.3049550195428310	0.00019171741405443800	277457.3857730930	10000234.87850190
3	0.3049506209463290	0.00019722354772966400	277454.8044327500	10000234.00706860
4	0.3049316357464700	0.00019656596759887200	277452.6701963070	10000226.77146150
5	0.3049277808422630	0.00019318674821988700	277453.5076987460	10000224.92607500
6	0.3049647134103000	0.00019982519188488400	277455.8009158180	10000240.00340460
7	0.3049555104989850	0.00020031353778904300	277454.1853681500	10000236.48560330
8	0.3049548990748010	0.00019978716136392900	277454.4488261540	10000236.17024040
9	0.3049288106703900	0.00020056617086083900	277450.8456939540	10000226.08642580
10	0.3049286194000160	0.00020166417198197500	277450.3509142410	10000226.13976670
11	0.3049653218658930	0.00020770193623320700	277452.7553307410	10000241.70980830
12	0.3049564191460380	0.00020765534281963500	277451.3919835390	10000238.06964870
13	0.3049507642317620	0.00020415608651092000	277452.0944607560	10000235.28632740
14	0.3049336240219420	0.00020523982402664800	277449.5091442170	10000228.49348830
15	0.3049283350264890	0.00021033161283412500	277446.8513877650	10000226.93088720
16	0.3049658105373960	0.00021690113862860000	277448.9929368500	10000243.60369680
17	0.3049566075787880	0.00021654903866874500	277447.7120940880	10000239.60727690
18	0.3049469828074510	0.00021745946742157700	277445.9801086280	10000235.62153050
19	0.3049386181446610	0.00021771459250885500	277444.8541029100	10000232.08240890
20	0.3049283390901110	0.00021820514211867700	277443.5554540750	10000227.75148390
21	0.3049589044421740	0.00021580488237305000	277448.3121968660	10000240.37872120
22	0.3049564940483830	0.00022431032084568900	277444.3174804610	10000240.83905790
23	0.3049475903280840	0.00022486765192297800	277442.8339265640	10000236.96169090
24	0.3049389900188540	0.00022619678657065400	277441.1985277320	10000233.30735780
25	0.3049309250527590	0.00022781945244787500	277439.6384934190	10000229.88122370

UTM (E) equals aX-bY+Tx (3.1)

UTM (N) equals bX+aY+Ty (3.2)

Table 4: UTM TO Cassini Transformation Parameters for Limuru Sheet 148/1

UTM TO CASSINI TRANSFORMATION PARAMETERS				
1	**a**	**b**	**Tx**	**Ty**
2	3.279075096710590	-0.002055427912637240	-930363.5718326150	-32790961.42236330
3	3.279170834226530	-0.002061530758510340	-930445.9583870170	-32791906.56103520
4	3.279218050505730	-0.002120799152180550	-931043.2909221090	-32792359.43945310
5	3.279422229767080	-0.002113991243277270	-931024.8459744260	-32794379.44384770
6	3.279463732469590	-0.002077701676171270	-930676.2000276210	-32794798.49707030
7	3.279066474467980	-0.002148576721992870	-931282.298313750	-32790855.59765620
8	3.279165431362340	-0.002153957496375370	-931358.2580566410	-32791832.16455080
9	3.279172011505580	-0.002148306060917090	-931304.4311523440	-32791898.50097660
10	3.279452558577760	-0.002157051913854960	-931457.8922639690	-32794668.54931640
11	3.279454594798150	-0.002168863361475810	-931574.9523683110	-32794685.81201170
12	3.279059823194980	-0.002233260748653270	-932117.3305664060	-32790771.18896480
13	3.279155558440830	-0.002232890093182500	-932135.7067143180	-32791716.732910200
14	3.279216414375700	-0.002195344486608520	-931779.4237833890	-32792326.59570310
15	3.279400723928120	-0.002207246380748980	-931941.0892333980	-32794144.16162110
16	3.279457533557430	-0.002262084286485330	-932496.5229517250	-32794691.93750000
17	3.279054437065490	-0.002332165155166880	-933092.57104492200	-32790696.10351560
18	3.279153396491890	-0.002328519972252250	-933079.3650265570	-32791673.63183590
19	3.279256885522050	-0.002338457151381590	-933201.7628173830	-32792692.7221680
20	3.279346825554960	-0.002341329131013480	-933251.7365722660	-32793579.74414060
21	3.279457373311740	-0.002346762740671690	-933332.4753961410	-32794669.53955080
22	3.279128707523340	-0.002320483103176230	-932994.1143685670	-32791431.49426270
23	3.279154499352440	-0.002411977541669330	-933903.1381441760	-32791665.55224610
24	3.279250223829880	-0.002418111504084660	-933986.1637643370	-32792608.40917970
25	3.279342688270840	-0.002432541629787010	-934150.7500319980	-32793517.09130860
1	3.279429398418870	-0.002450121508445590	-934345.4852334250	-32794368.103515600

CASSINI (E) equals aX-bY+Tx (3.12)

CASSINI (N) equals bX+aY+Ty (3.13)

4.5.2. Plug in Specifications

The plug in will have the following specifications:

1. The plug in will read coordinates from an input file (shapefile - points)
2. The plug in will then determine the coordinate system of the input shapefile (either UTM or CASSINI)
3. The plug in will identify the type of conversion (either UTM to CASSINI or CASSINI to UTM) based on the input coordinate system
4. The plug in will identify the range within which the input coordinates fall, that is the grid corners (contained in a database)
5. The plug in will then assign apropriate transformation parameters (contained in a database) that apply to the identified grid corners, see table 3 and 4
6. The plug in will then compute the transformations using the assigned transformation parameters (contained in a database)
7. The plug in will finally compile the transformed coordinates into a new file (shapefile - points)

4.5.3. The plug in Components

1. __init__.py

This is the starting point of the plugin. This file is essential in Python's import system and must have the **classFactory()** method which is called when the plug-in gets loaded to QGIS. It gets reference to instance of **QgisInterface** and must return instance of the plugin's class from the mainplugin.py. The file may have any other initialization code (QGIS, 2017).

2. utm_cassini_converter_module (mainPlugin.py)

This is the main working code of the plugin. It contains all the information about the actions of the plugin and the main code (QGIS, 2017).

3. metadata.txt

This is a .xml document created by Qt Designer. The file contains relative paths to resources of the forms. The plugin manager retrieves some information about the plugin such as its name, description etc from this file. (QGIS, 2017).

4. resources.qrc

This the translation of the .qrc file described above to Python (QGIS, 2017).

5. ***resources.py***

This is the GUI created by Qt Designer (QGIS, 2017).

6. ***Database***

Four csv files act as the plug in database. These are:

1. CASSINI_identify_sheet_no;
2. CASSINI_to_UTM_with_sheet_no;
3. UTM_identify_sheet_no; and
4. UTM_to_CASSINI_with_sheet_no

`CASSINI_identify_sheet_no` is a csv file that contains all the Cassini sheet corners for a perticular Topographic map. These are used to identify the grid extent within which a Cassini input coordinate fall. This is important in assigning transformation parameters that apply to the grid within which the input point falls; when converting from Cassini to UTM.

`CASSINI_to_UTM_with_sheet_no` is a csv file containing all the transformation parameters that facilitate Cassini to UTM conversion.

`UTM_identify_sheet_no` is a csv file containing all the UTM sheet corners for a perticular Topographic map. These are used to identify the grid extent within which the input UTM coordinate fall. These are used in assigning transformation parameters that apply to the grid within which the input point falls; when converting from UTM to Cassini.

`UTM_to_CASSINI_with_sheet_no` is a csv file that contains all the transformation parameters that facilitate UTM to Cassini conversion.

4.5.4. Plug in Work Flow

The plug in works as shown in the flow chart diagram below. The software requres an input, which is a Shapefile-point. Once the input is selected, the software will automatically determine its coordinate system and hence the type of transformation (whether Cassini to UTM or UTM to Cassini). Once that has been determined, the software determines whether the points are within the database range. It then assigns the appropriate transformation parameters that apply to each of the coordinates before computing the transformation. If the output location is included, the transformed coordinates are is saved within the perticular location. If the output location is not selected, the transformation will not be computed until the output location is specified, see figure 9 and 10. The Python code for the entire process is attached on appendix C.

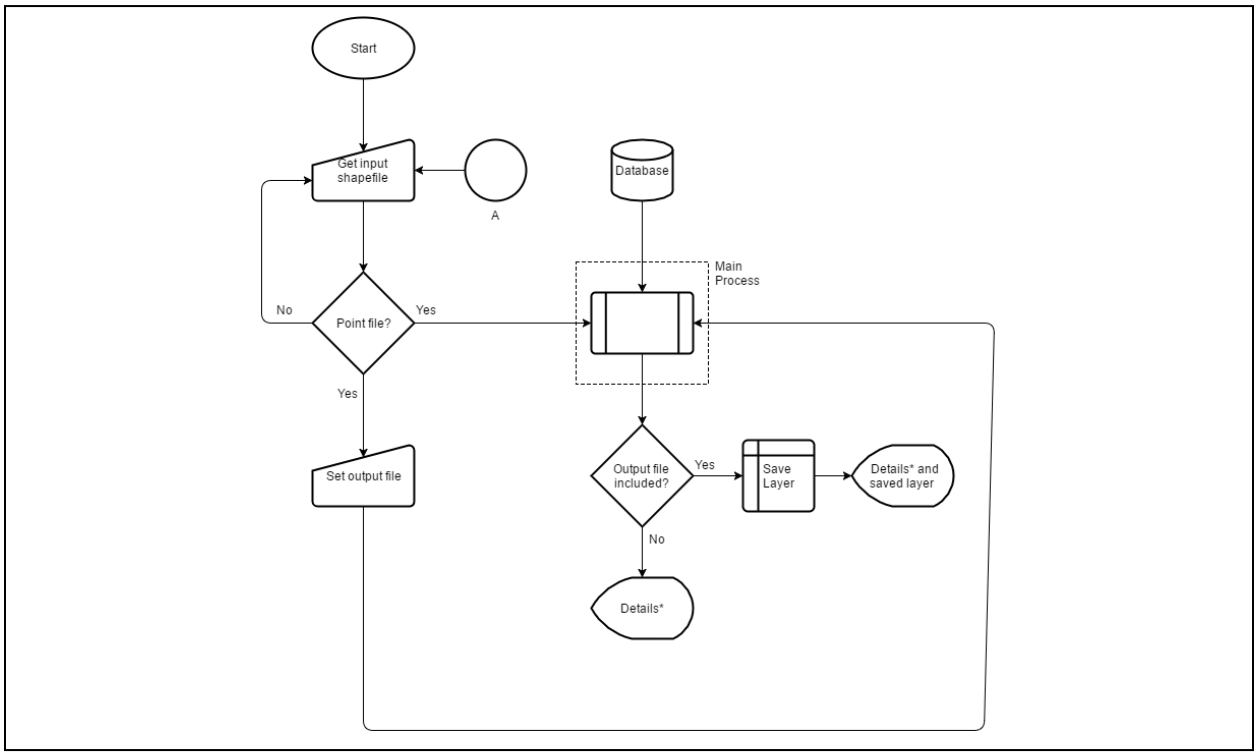


Figure 10: Process of converting coordinates between Cassini and UTM as in the proposed Plug

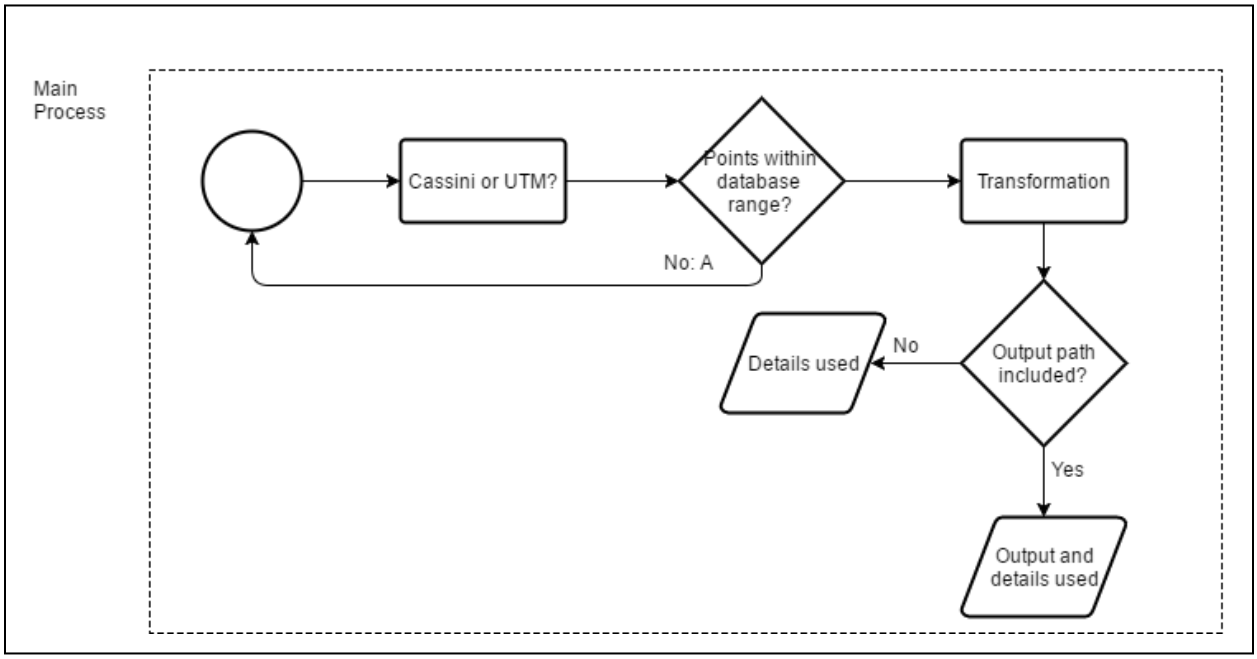


Figure 11: Main Process; see figure 9

4.5.5. Testing the QGIS UTM Cassini Inter-Converter

The plug is installed by copying the 'utm cassini converter class' folder into the to the QGIS python plugin directory 'c:\Users\username\.qgis2\python\plugins'. This is because the plug in has not yet been published. Once published, it will be possible to install the plug in directly using the plug in installer.

After copying the 'utm cassini converter class' folder into the specified folder, the plug-in icon will appear under vector – UTM CASSINI inter converter – UTM Cassini Converter. Once you click on UTM-Cassini Converter, the plug in window appears, see figure 12. Figure 13 shows the named parts of the plug in window.

The plug-in only works for Cassini and UTM coordinates within the range covered by Topograhic Map Sheet 148/1 for Limuru.

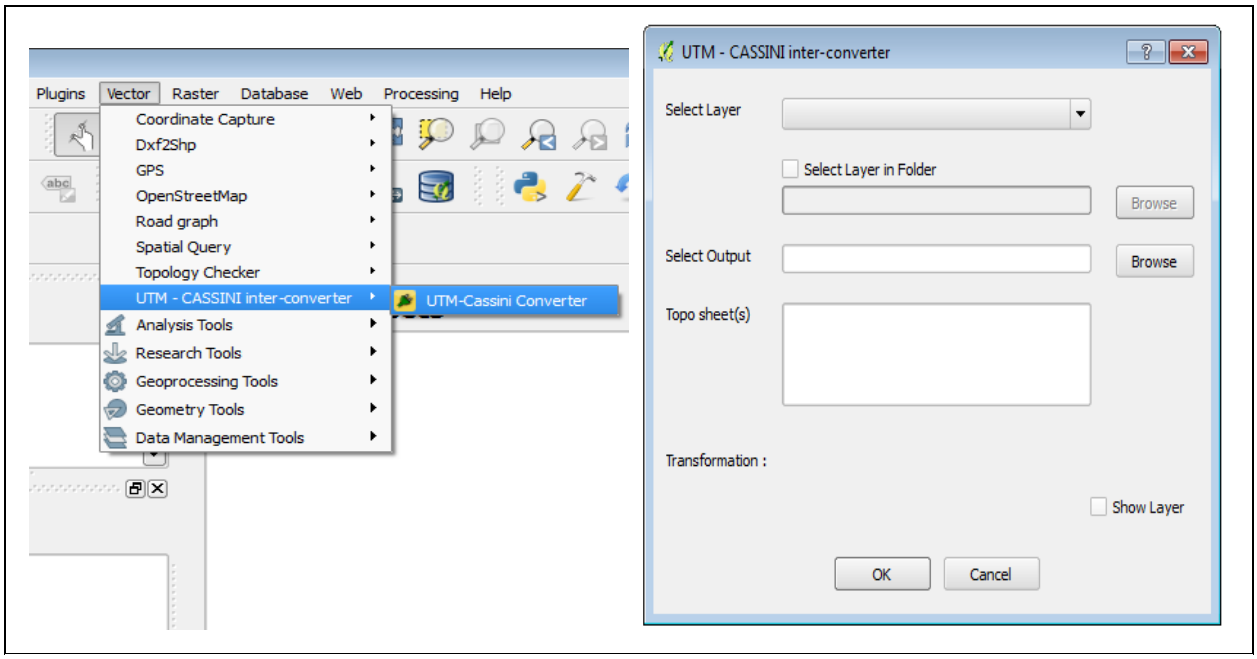


Figure 12: How to access the UTM Cassini Converter

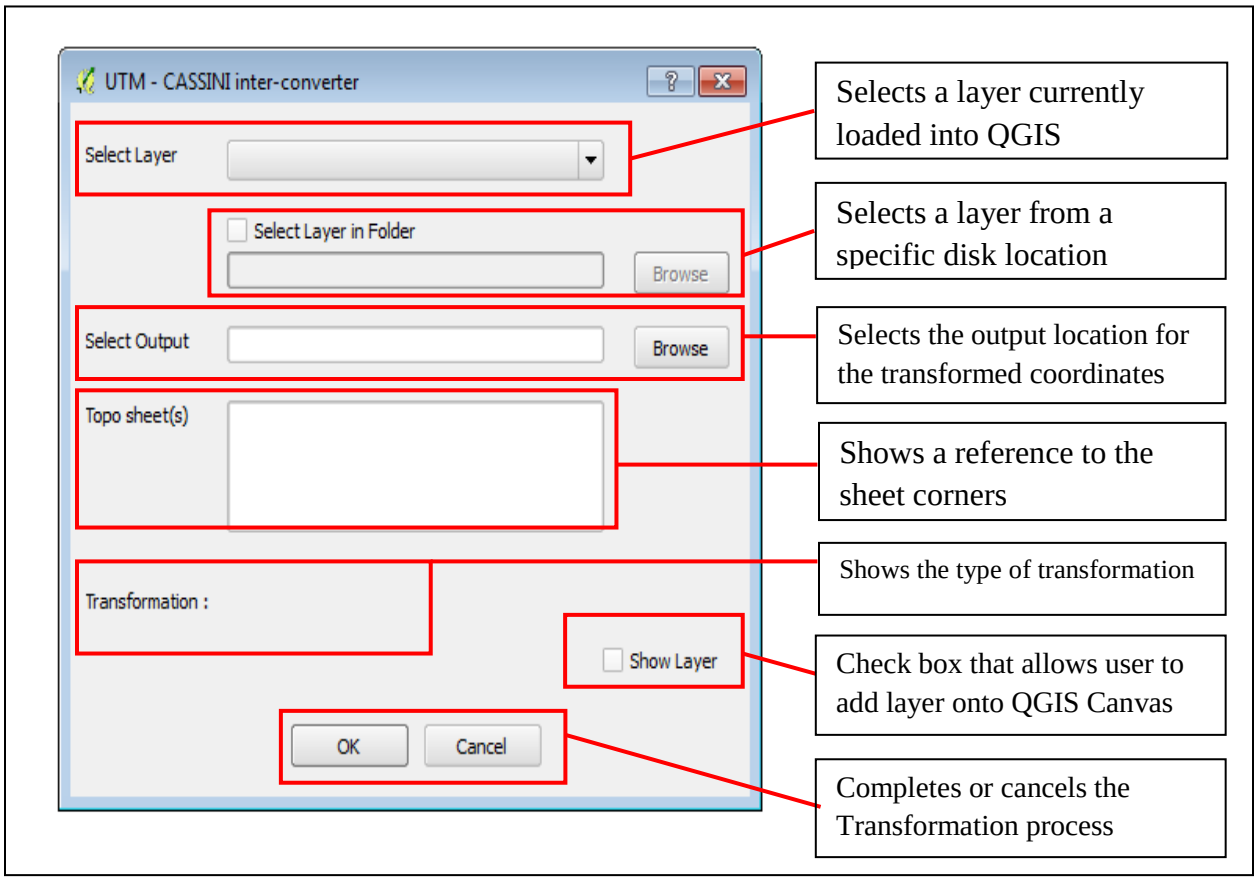


Figure 13: The UTM Cassini Inter Converter User Interface

4.5.6. Accuracy assessment

Two separate sets of coordinates in UTM and Cassini were converted and the percentage error between the original points and converted points calculated as shown in table 5 and 6. The mean percentage error when converting from Cassini to UTM is 0.000 for both X and Y coordinates. The error is 0.0001 for X and 0.0000 for Y when converting UTM to Cassini.

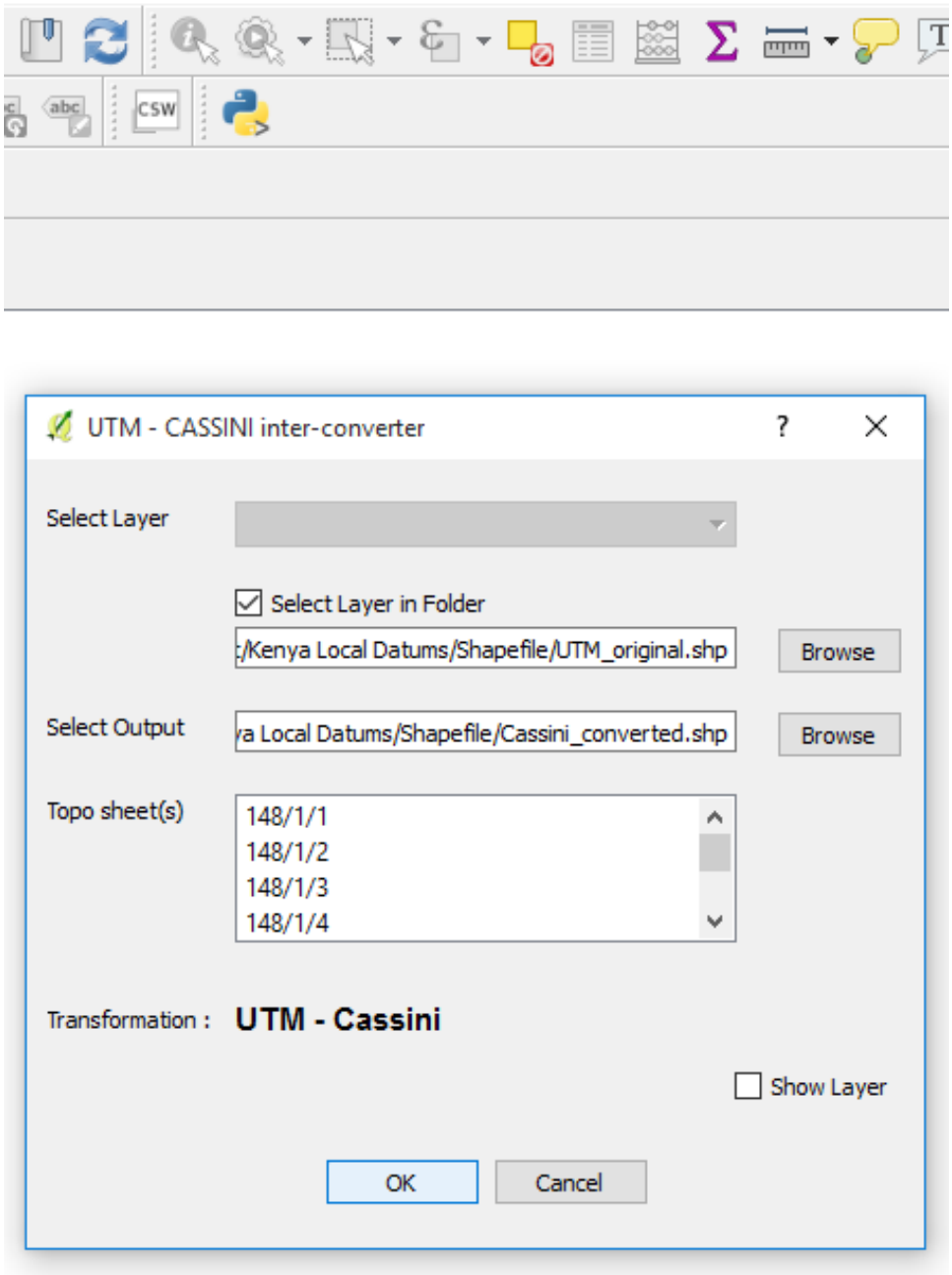


Figure 14: Converting coordinates from UTM to Cassini in order to calculate the error

Table 5: Cassini to UTM Error Calculation

ORIGINAL UTM		CONVERTED UTM		ERROR		% ERROR	
UNITS: METER							
POINT_X	POINT_Y	POINT_X	POINT_Y	POINT_X	POINT_Y	POINT_X	POINT_Y
224547.2252	9886610.4754	224547.1846	9886610.4707	0.0406	0.0047	0.0000	0.0000
230115.3254	9886614.7752	230115.2902	9886614.7711	0.0352	0.0041	0.0000	0.0000
235683.2005	9886618.9503	235683.1636	9886618.8719	0.0370	0.0784	0.0000	0.0000
241250.8503	9886623.1001	241250.8076	9886623.1817	0.0427	0.0817	0.0000	0.0000
246818.3503	9886627.1001	246818.3154	9886627.0963	0.0349	0.0038	0.0000	0.0000
224551.6254	9881079.2753	224551.5904	9881079.2665	0.0350	0.0087	0.0000	0.0000
230119.6003	9881083.7501	230119.5580	9881083.7545	0.0424	0.0044	0.0000	0.0000
235687.3754	9881088.1252	235687.3480	9881088.1347	0.0274	0.0094	0.0000	0.0000
241254.9752	9881092.4750	241254.9346	9881092.4633	0.0406	0.0117	0.0000	0.0000
246822.4504	9881096.6004	246822.4077	9881096.6050	0.0427	0.0046	0.0000	0.0000
224556.2254	9875548.0753	224556.1827	9875548.0774	0.0427	0.0022	0.0000	0.0000
230124.1256	9875552.7255	230124.0960	9875552.7225	0.0296	0.0030	0.0000	0.0000
235691.7502	9875557.3502	235691.7020	9875557.4242	0.0482	0.0740	0.0000	0.0000
241259.3254	9875561.8252	241259.2905	9875561.7405	0.0349	0.0847	0.0000	0.0000
246826.7005	9875566.2003	246826.6636	9875566.2014	0.0370	0.0011	0.0000	0.0000
224561.0505	9870016.9003	224561.0132	9870016.8909	0.0373	0.0094	0.0000	0.0000
230128.7752	9870021.8250	230128.7346	9870021.8275	0.0406	0.0025	0.0000	0.0000
235696.4003	9870026.6001	235696.3651	9870026.5985	0.0352	0.0016	0.0000	0.0000
241263.8255	9870031.1756	241263.7829	9870031.1719	0.0425	0.0037	0.0000	0.0000
246831.0503	9870035.8501	246831.0153	9870035.8486	0.0350	0.0015	0.0000	0.0000
230133.7504	9864490.8004	230133.5426	9864490.5953	0.2078	0.2051	0.0001	0.0000
235701.2504	9864495.7504	235701.2156	9864495.7378	0.0348	0.0126	0.0000	0.0000
241268.5755	9864500.6751	241268.5384	9864500.6777	0.0371	0.0025	0.0000	0.0000
246835.6253	9864505.4254	246835.5922	9864505.3905	0.0330	0.0348	0.0000	0.0000
					TOTAL	0.0005	0.0000
					MEAN % ERROR	0.0000	0.0000

Table 6: UTM to Cassini Error Calculation

FID	ORIGINAL CASSINI		CONVERTED CASSINI		ERROR		% ERROR	
	UNITS: FEET							
	POINT_X	POINT_Y	POINT_X	POINT_Y	POINT_X	POINT_Y	POINT_X	POINT_Y
0	-173735.2753	-372484.7739	-173735.1414	-372484.7571	0.1339	0.0167	0.0001	0.0000
1	-155477.0496	-372482.1493	-155476.9321	-372482.1319	0.1175	0.0174	0.0001	0.0000
2	-137219.2741	-372480.2227	-137219.1538	-372479.9516	0.1203	0.2711	0.0001	0.0001
3	-118961.3499	-372477.6018	-118961.2122	-372477.8681	0.1377	0.2663	0.0001	0.0001
4	-100703.0247	-372476.3248	-100702.9068	-372476.3085	0.1179	0.0163	0.0001	0.0000
5	-173732.4496	-390622.2993	-173732.3322	-390622.2881	0.1174	0.0113	0.0001	0.0000
6	-155474.7248	-390619.5750	-155474.5902	-390619.5736	0.1347	0.0013	0.0001	0.0000
7	-137217.4745	-390617.1747	-137217.3854	-390617.2002	0.0891	0.0255	0.0001	0.0000
8	-118959.7503	-390614.9997	-118959.6163	-390614.9465	0.1340	0.0531	0.0001	0.0000
9	-100701.3248	-390613.4740	-100701.1884	-390613.4842	0.1364	0.0102	0.0001	0.0000
10	-173729.4998	-408759.7495	-173729.3576	-408759.7590	0.1422	0.0095	0.0001	0.0000
11	-155471.9740	-408756.9738	-155471.8783	-408756.9639	0.0956	0.0099	0.0001	0.0000
12	-137215.0755	-408754.0260	-137214.9223	-408754.2643	0.1532	0.2383	0.0001	0.0001
13	-118957.3996	-408752.3477	-118957.2886	-408752.0791	0.1110	0.2686	0.0001	0.0001
14	-100699.5992	-408750.2995	-100699.4800	-408750.3047	0.1191	0.0052	0.0001	0.0000
15	-100698.0497	-426886.9500	-100697.9326	-426886.9537	0.1171	0.0037	0.0001	0.0000
16	-118955.1248	-426889.2233	-118954.9861	-426889.2098	0.1386	0.0135	0.0001	0.0000
17	-137212.1997	-426891.2000	-137212.0862	-426891.1926	0.1135	0.0074	0.0001	0.0000
18	-155469.4003	-426893.9000	-155469.2672	-426893.9033	0.1332	0.0033	0.0001	0.0000
19	-173726.2741	-426897.1243	-173726.1539	-426897.1032	0.1203	0.0211	0.0001	0.0000
20	-155466.2248	-445031.2240	-155465.5435	-445030.5380	0.6813	0.6860	0.0004	0.0002
21	-137209.4496	-445028.4987	-137209.3300	-445028.4628	0.1196	0.0359	0.0001	0.0000
22	-118952.7242	-445025.8250	-118952.5989	-445025.8262	0.1253	0.0012	0.0001	0.0000
23	-100696.3501	-445023.8982	-100696.2375	-445023.7887	0.1126	0.1095	0.0001	0.0000
						TOTAL	0.0026	0.0005
						MEAN % ERROR	0.0001	0.0000

CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

All the three objectives of the study have been adequately addressed as outlined in the report. The methods used by the Kenyan GIS Community to convert coordinates between Cassini and UTM including their associated challenges were established and reviewed. The solutions to these challenges were formulated; and these solutions formed the basis for the design of a QGIS helper application that converts coordinates between Cassini to UTM coordinate systems. The application developed was installed and tested on different computer platforms with satisfactory results obtained during the conversions.

The results obtained by using the application were as accurate as the ones obtained using the excel spreadsheet as demonstrated in the report. This method has a major advantage over all the other methods in that; the user does not need to have the sheet corners that apply to the points to be transformed; and the user does not need to keep on keying in data (sheet corners) so as to compute transformation parameters. All these parameters are contained in a database which is part of the software. Thus, this method easily solves the problem of converting coordinates between Cassini and UTM in Kenya by presenting a uniform way for every user free of human error.

5.2 Recommendations

The program demonstrated in this report works for coordinates that fall within geographical coordinates 36.5°,-1.0° to 36.75°,-1.25°. This is the area covered under the Limuru Topo Sheet 148/1. The functionality of the application can be extended easily for the entire country by populating the database containing the transformation parameters to include those for the rest of the Country. For areas where topo-sheets do not have sheet corners, surveys are needed in order to obtain Cassini and UTM sheet corners.

Figure 15 shows the available sheet corners for the whole Country. The plug in can be implemented for the whole country as follows:

5.2.1 Phase 1: Plug in design and piloting

Phase 1 involves designing the plug in and piloting it on extents under geographical coordinates 36.5°, -1.0° to 36.75°, -1.25° south; Limuru Topo Sheet 148/1. This phase is already complete and successful as presented in this report. The plug in has been designed and tested successfully for this area.

5.2.2 Phase 2: Plug in implementation on areas with sheet corners

Phase 2 involves extending the functionality of the plug in to include areas that currently have sheet corners that facilitate the Cassini and UTM transformation. These will be sourced from Survey of Kenya. The process includes creating a database containing transformation parameters calculated using these sheet corners. This database will be added to the main plug in database. The areas that have available sheet corners are shown on figure 15. The plug in will be published at this stage for public use.

5.2.3 Phase 3: Plug in implementation for the remaining areas of the Country

The final phase will involve finalizing the plug in database such that it contains all the transformation parameters for converting between Cassini and UTM for the whole country. This means obtaining transformation parameters for all the areas that do not have sheet corners. Figure 15 shows the Topo Maps in Kenya in which the sheet corners are not available.

For these areas that do not have transformation parameters, a GPS Survey is necessary. This will be done by using Cassini points in cadastral maps to obtain their corresponding UTM points using a GPS. These two sets of points will then be used to compute transformation parameters which will be populated in the plug-in database. The plug in at this final stage will be updated such that it will be available to the public.

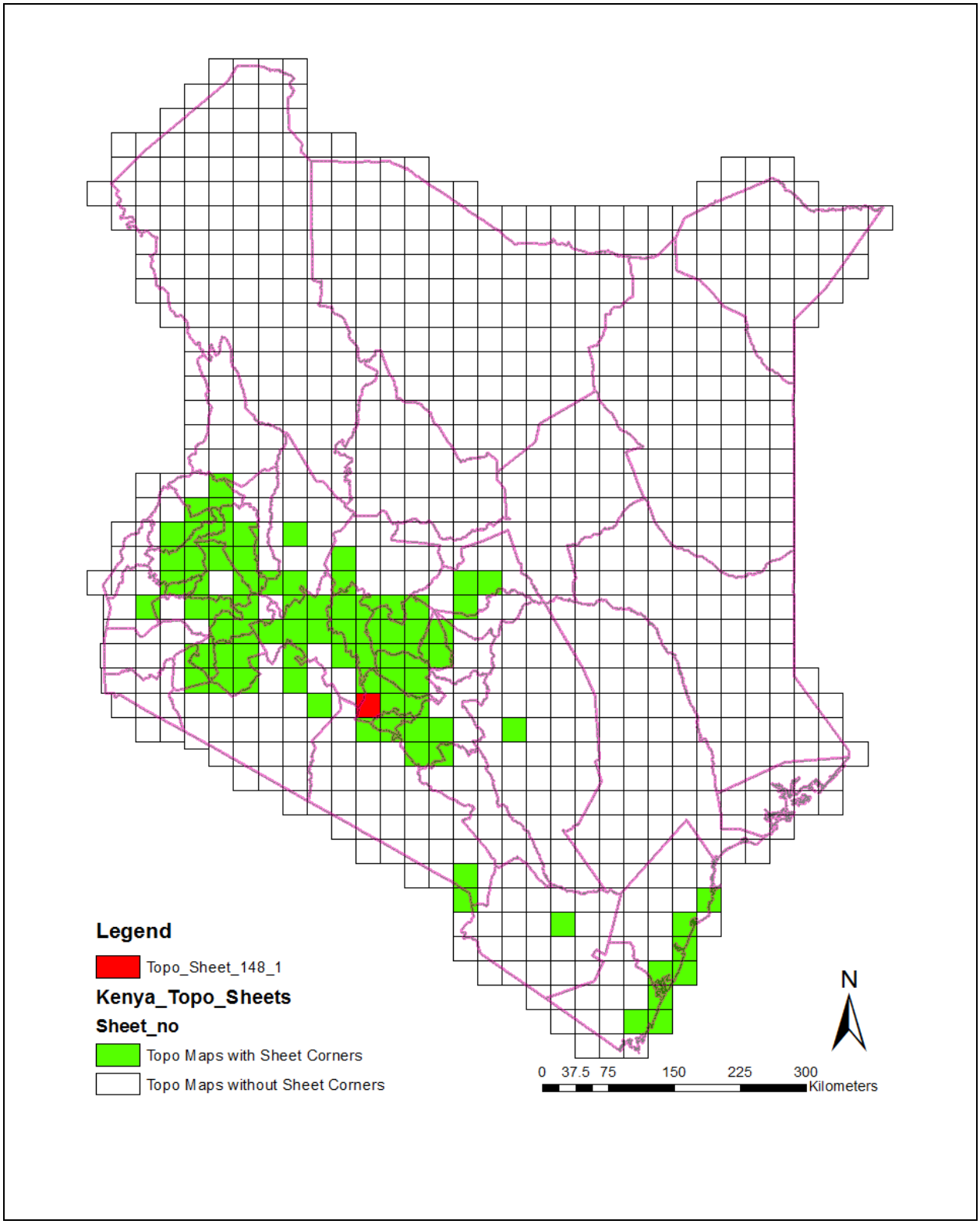


Figure 15: Available Sheet Corners for Kenya

REFERENCES

Briggs, R. (1998, October 15). GIS Software & Hardware Overview; POEC Introduction to GIS; POEC 6383 GIS Implementation & Management. Dallas, United States of America.

Caitlin Dempsey. (2017, April 28). Learning GIS Programming. Retrieved from GIS Launge: https://www.gislounge.com/learning-programming-for-gis/

DigiKom Ltd. (2014). User's Guide for SurveyingCalculation plugin for QGIS 2.x. Retrieved June 27, 2017, from http://www.digikom.hu/SurveyingCalculation/usersguide.html

ESRI. (2004). ArcGIS 9; what is ArcGIS? New York: ESRI.

ESRI. (2004). Understanding Map Projections.

ESRI. (2017). What is Python? Retrieved from ESRI: http://desktop.arcgis.com/en/arcmap/10.3/analyze/python/what-is-python-.htm

Gábor Timár, G. M. (2013). Map grids and datums. Eötvös Lóránd University.

Gachoki T.G., F. A. (2013). Transformation between GPS Coordinates and Local Plane UTM Coordinates using the Excel Spreadsheet. Nairobi.

Gachoki, T. (2013). Conversion of Cassini Coordinates To UTM Coordinates on The Excel Spreadsheet. Nairobi: KENHA.

Gachoki, T. G. (2013). Conversion of UTM Coordinates to Cassini Coordinates on the Excel Spread Sheet. Nairobi: KENHA.

GIS Geography. (2017, September 13). 27 Differences between ArcGIS and QGIS – The Most Epic GIS Software Battle in GIS History. Retrieved from GIS Geography: http://gisgeography.com/qgis-arcgis-differences/

Kamau, M. (2016, February 11). The current geodetic reference system in Kenya. Retrieved April 7, 2016, from WordPress.com: https://edembac.wordpress.com/2012/05/26/the-current-geodetic-reference-system-in-kenya/

Kennedy, M. (2000). Understanding Map Projections. New York: ESRI.

Mugnier, C. (2000). Geodetic Report of Kenya. Louisiana State University: Research Gate.

MyGeodata Cloud. (2016). Coordinate system transformation of value pairs on-line (cs2cs. Retrieved June 27, 2017, from https://mygeodata.cloud/cs2cs/

QGIS. (2017, July 3). Developing Python Plug ins. Retrieved July 3, 2017, from http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/plugins.html

QGIS. (2017, December 4). QGIS. Retrieved from QGIS: https://www.qgis.org/en/site/

QGIS Project. (2014). PyQGIS developer cookbook; Release 2.2.

Satya Prakash Maurya, A. O. (2015). Open Source GIS: A Review. Varanasi: ReearchGate.

Snyder, J. P. (1987). Map Projections - A Working Manual; US Geologica Survey Professional Paper 1395. Washington: United States Government Printing Office.

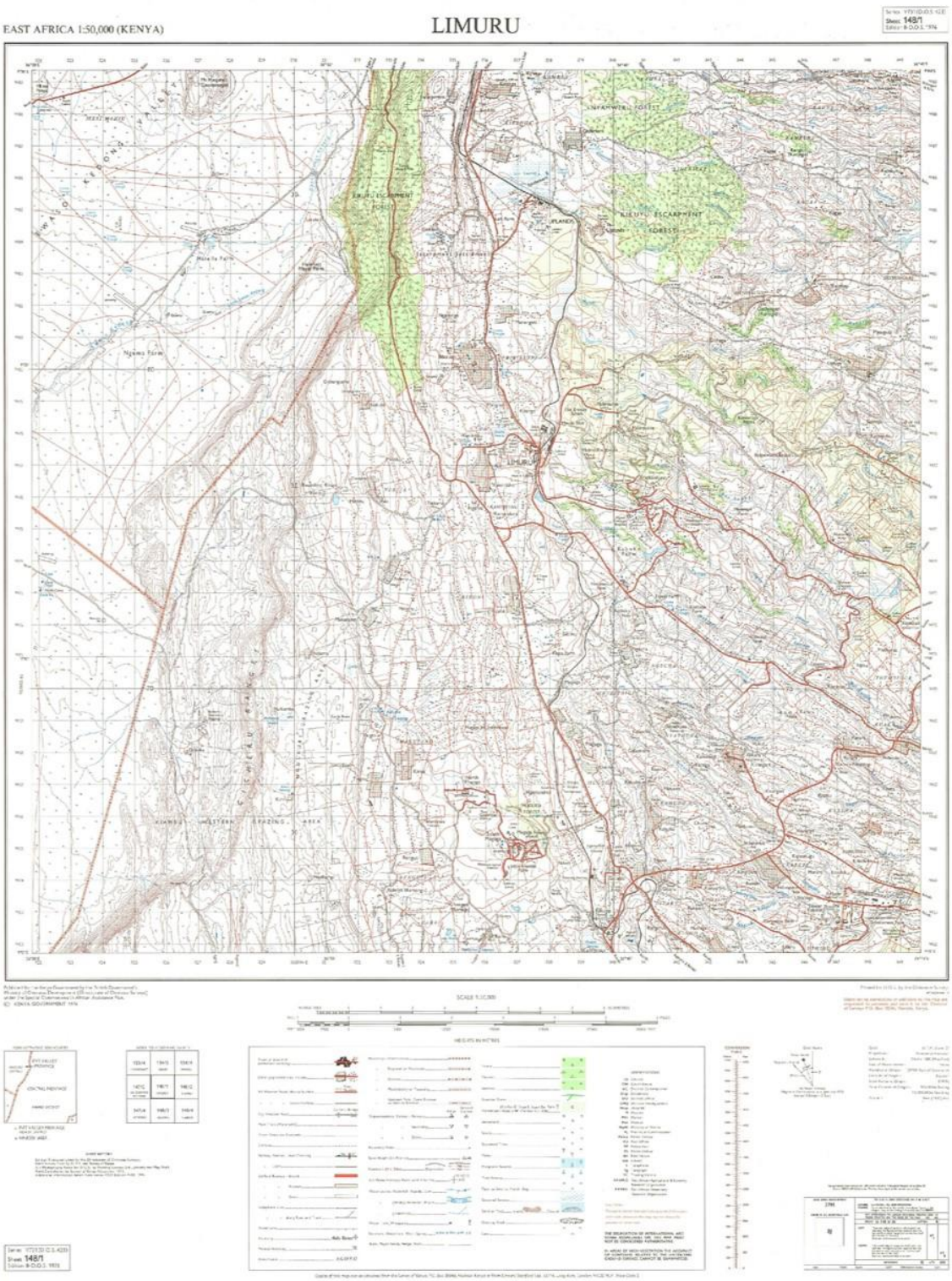
The Pennsylvania State University. (2017). Overview of Programming Languages for GIS. Retrieved from PennState Colledge of Earth and Mineral Sciences; Department of Geography: https://www.e-education.psu.edu/geog583/node/67

Thomas Soler, L. D. (1989). Important Parameters Used in Geodetic Transformations.

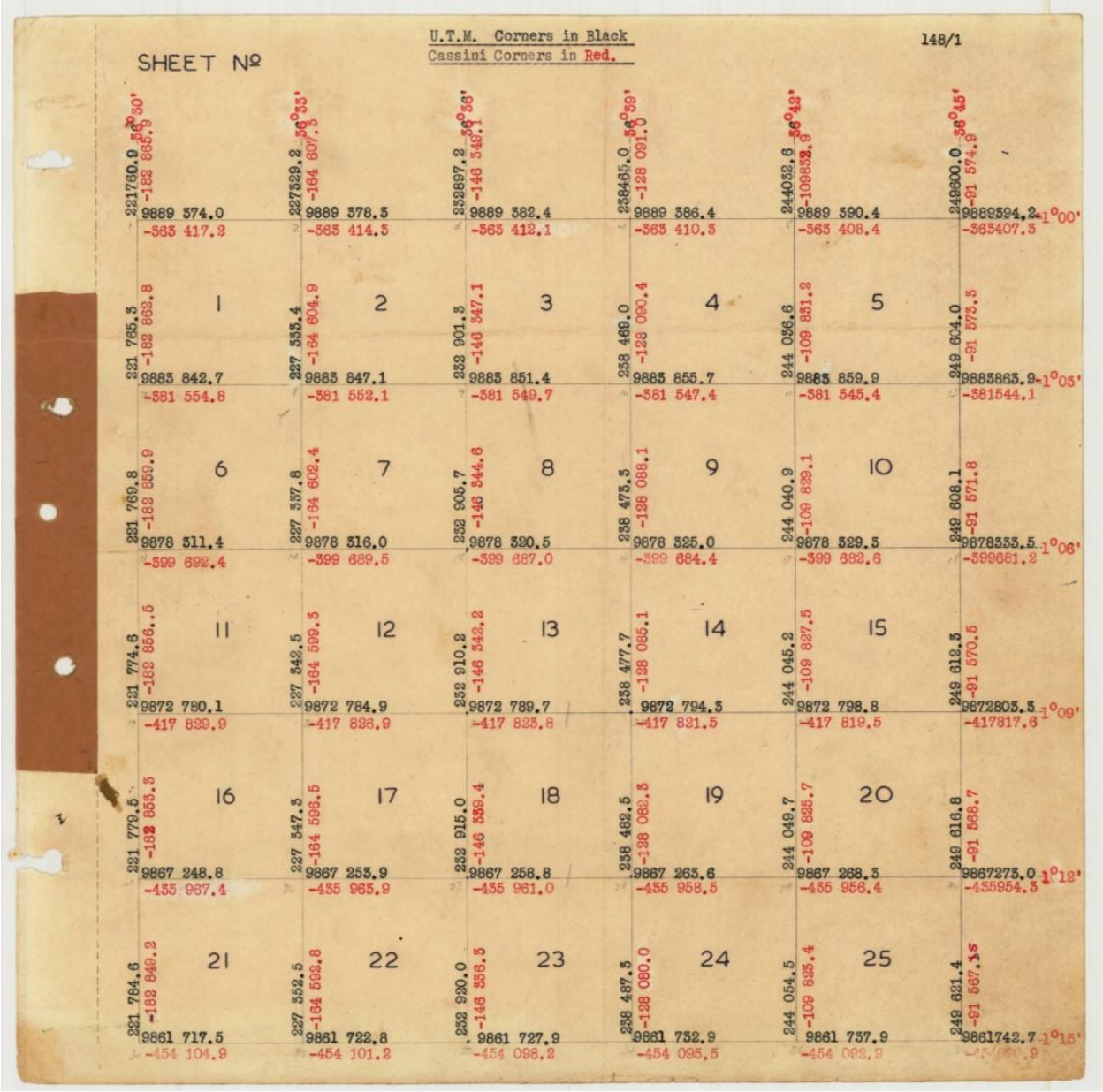
Torge, W. (2001). Geodesy. Berlin: Walter de Gruyter.

APPENDICES

Appendix A: Limuru Topo sheet 148/1



Appendix B: Limuru Topo sheet 148/1 Sheet Corners



Appendix C: Plug-in Python code

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 utm_cassini_converter_class
                                 A QGIS plugin
 Transforms UTM to CASSINI and vice-versa
                             -------------------
        begin                : 2017-06-19
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Dissent Ingati
        email                : ingatid@gmail.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QTextStream,
QCoreApplication, QFileInfo
from PyQt4.QtGui import QAction, QIcon, QFileDialog
from qgis.core import QGis, QgsMessageLog, QgsMapLayer
from osgeo import ogr
# Initialize Qt resources from file resources.py
import resources_rc
# Import the code for the dialog
from utm_cassini_converter_module_dialog import
utm_cassini_converter_classDialog
import os.path
import csv
from osgeo import ogr, gdal
import osgeo.osr as osr
import json, os
from qgis.gui import QgsMessageBar
from os.path import expanduser
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from qgis.core import *
from qgis.gui import *

#DI - Feed the CSV files to the lists in the plugin

home = expanduser("~")

filep = home + "\\.qgis2\\python\\plugins\\utm_cassini_converter_class\\"

Cassini_SheetNo_Id = "CASSINI_identify_sheet_no.csv"
UTM_SheetNo_Id = "UTM_identify_sheet_no.csv"
Cassini_UTM_Params = "CASSINI_to_UTM_with_sheetno.csv"
UTM_Cassini_Params = "UTM_to_CASSINI_with_sheetno.csv"

all_csv = [Cassini_SheetNo_Id, UTM_SheetNo_Id, Cassini_UTM_Params,
UTM_Cassini_Params]
C_S_I = []
U_S_I = []
C_U_P = []
U_C_P = []

for fyl in all_csv:
    with open(filep + fyl, 'rb') as f:
```

```
        reader = csv.reader(f)
        your_list = list(reader)

        for k in your_list:
            if your_list[0] != k and fyl == Cassini_SheetNo_Id:
                C_S_I.append(k)
            if your_list[0] != k and fyl == UTM_SheetNo_Id:
                U_S_I.append(k)
            if your_list[0] != k and fyl == Cassini_UTM_Params:
                C_U_P.append(k)
            if your_list[0] != k and fyl == UTM_Cassini_Params:
                U_C_P.append(k)

new_coords = []
ref = ""
shp_name = ""
oncanvas = ""
out_name = ""
oncanvas = False
cas_e = ""
cas_n = ""
sn = ""
utm_e = ""
utm_n = ""

class utm_cassini_converter_class:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'utm_cassini_converter_class_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&UTM - CASSINI inter-converter')
        # TODO: We are going to let the user set this up in a future
iteration
        self.toolbar = self.iface.addToolBar(u'utm_cassini_converter_class')
        self.toolbar.setObjectName(u'utm_cassini_converter_class')

        # Create the dialog (after translation) and keep reference
        self.dlg = utm_cassini_converter_classDialog()
```

```
        #DI - Give the widget tools in the interface functionality on any
changes made by user
        self.dlg.txtOutput.clear()
        self.dlg.btnOutput.clicked.connect(self.select_output_file)

        self.dlg.chkSelect.stateChanged.connect(self.state_changed_Select)
        self.dlg.chkLoad.stateChanged.connect(self.state_changed_Load)

        self.dlg.cboLayer.currentIndexChanged.connect(self.cbo_state_changed)

        self.dlg.txtInput.clear()
        self.dlg.btnInput.clicked.connect(self.select_input_file)

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('utm_cassini_converter_class',
message)

    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.

        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this action.
        :type text: str

        :param callback: Function to be called when the action is triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be
enabled
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
```

```
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse
pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is also
            added to self.actions list.
        :rtype: QAction
        """
        # Create the dialog (after translation) and keep reference
        #self.dlg = utm_cassini_converter_classDialog()

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            self.toolbar.addAction(action)

        if add_to_menu:
            self.iface.addPluginToVectorMenu(
                self.menu,
                action)
        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/utm_cassini_converter_class/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'UTM-Cassini Converter'),
            callback=self.run,
            parent=self.iface.mainWindow())

    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""

        for action in self.actions:
            self.iface.removePluginVectorMenu(
                self.tr(u'&UTM - CASSINI inter-converter'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar
    def run(self):
```

```
        global layers
        """Run method that performs all the real work"""
        #DI - Get all point layers on layer panel and add to Combo box
      layers = self.iface.legendInterface().layers()
      layer_list = []
      for layer in layers:
            if layer.type() == QgsMapLayer.VectorLayer and
layer.geometryType() == QGis.Point:
                layer_list.append(layer.name())
      self.dlg.cboLayer.clear()
      self.dlg.cboLayer.addItems(layer_list)

      #self.dlg.txtOutput.clear()
        #self.dlg.btnOutput.clicked.connect(self.select_output_file)

      # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            #DI - get the processed input and output names including spatial
reference if any,
            #and the processed coordinates. oncanvas is a boolean value which
indicates map layer to be displayed if true
            out_name, new_coords, shp_name, ref, oncanvas =
self.check_projection_stuff()
            # Do something useful here - delete the line containing pass and
            # substitute with your code.
            #DI - If the input and output files are valid and the processed
coordinates are found, proceed
            if out_name != "" and new_coords != [] and shp_name != "":

                #DI - The following illustrates the OGR vector creation
process
                #Driver - shows what we will use to create a shapefile in
this case
                driver = ogr.GetDriverByName("ESRI Shapefile")
                data_source = driver.CreateDataSource(out_name)
                lyrname = out_name.split("/")[-1:][0][:-4].encode('utf-8')

                layerx = data_source.CreateLayer(lyrname, None, ogr.wkbPoint)
                #DI - after creating an empty layer, we add field names and
their properties
                field_name = ogr.FieldDefn("Name", ogr.OFTString)
                field_name.SetWidth(10)
                layerx.CreateField(field_name)
                layerx.CreateField(ogr.FieldDefn("Northing", ogr.OFTReal))
                layerx.CreateField(ogr.FieldDefn("Easting", ogr.OFTReal))

                #DI - we fill up the fields and create a geometry for each
point in the layer
                n = 0
                for x in new_coords:
                    n = n + 1
                    feat_name = "Point_" + str(n)
                    feature = ogr.Feature(layerx.GetLayerDefn())
                    feature.SetField("Name", feat_name)
                    feature.SetField("Northing", x[1])
                    feature.SetField("Easting", x[0])

                    wkt = "POINT(%f %f)" %  (x[0], x[1])
                    point = ogr.CreateGeometryFromWkt(wkt)
                    feature.SetGeometry(point)
```

```
                    layerx.CreateFeature(feature)
                    feature = None

                data_source = None

                #DI - Add map to canvas if oncanvas is true
                if oncanvas:
                    layera = self.iface.addVectorLayer(out_name, lyrname,
"ogr")
                    if not layera:
                        self.iface.messageBar().pushMessage("Error", "Layer
failed to load!", level=QgsMessageBar.WARNING, duration=3)

            else:
                self.iface.messageBar().pushMessage("Error", "Something went
wrong", level=QgsMessageBar.WARNING, duration=3)
            self.iface.messageBar().pushMessage("Progress", "Conversion
completed", level=QgsMessageBar.INFO, duration=3)
            layer_list = []
            self.dlg.close()

    #DI - Get the output file path and add to line Edit field
    def select_output_file(self):

        filename_out = QFileDialog.getSaveFileName(self.dlg, "Specify output
file name ","", '*.shp')
        if filename_out:
            self.dlg.txtOutput.setText(filename_out)
    #DI - If browse button was used to select input file, disable the combo
box and vice versa
    def state_changed_Select(self, int):
        if self.dlg.chkSelect.isChecked():
            self.dlg.btnInput.setEnabled(True)
            self.dlg.txtInput.setEnabled(True)
            self.dlg.cboLayer.setEnabled(False)
            self.dlg.txtInput.clear()
        else:
            self.dlg.btnInput.setEnabled(False)
            self.dlg.txtInput.setEnabled(False)
            self.dlg.cboLayer.setEnabled(True)
            self.dlg.txtInput.clear()

    #DI - If output to be shown on map, oncanvas is true
    def state_changed_Load(self, int):
        global oncanvas
        if self.dlg.chkLoad.isChecked():
            oncanvas = True
        else:
            oncanvas = False

    #DI - Select the input shapefile via browse button
    def select_input_file(self):
        filename_in = QFileDialog.getOpenFileName(self.dlg, "Select input
file ","", '*.shp')

        vLayer = QgsVectorLayer( filename_in,
QFileInfo(filename_in).baseName(), "ogr" )
        layerGeometry = vLayer.geometryType()
        if filename_in and layerGeometry == QGis.Point:
            self.dlg.txtInput.setText(filename_in)
            out_name, new_coords, shp_name, ref, oncanvas =
self.check_projection_stuff()
        else:
```

```
            self.iface.messageBar().pushMessage("Input Type", "Please select
a point shapefile", level=QgsMessageBar.WARNING, duration=3)

    #DI - Check if a new layer has been selected in combo box
    def cbo_state_changed(self):
        out_name, new_coords, shp_name, ref, oncanvas =
self.check_projection_stuff()

    #DI - Calculates everything
    def check_projection_stuff(self):

        global new_coords
        global ref
        global shp_name
        global out_name
        global oncanvas

        self.dlg.lblTransformation.clear()
        self.dlg.lstToposheet.clear()
        new_coords = []
        ref = ""
        shp_name = ""
        oncanvas = False
        out_name = ""
        cas_e = ""
        cas_n = ""
        sn = ""
        utm_e = ""
        utm_n = ""
        yote_poa = False
        #DI - get input file path
        if self.dlg.chkSelect.isChecked():
            shp_name = self.dlg.txtInput.text()
        else:
            selectedLayerIndex = self.dlg.cboLayer.currentIndex()
            selectedLayer = layers[selectedLayerIndex]
            shp_name = selectedLayer.source()

        out_name = self.dlg.txtOutput.text()

        #DI - open the input file and get the spatial reference if any
        infile = ogr.Open(shp_name)
        layer = infile.GetLayer()
        ref = layer.GetSpatialRef()

        #DI - get the coordinates of each point in the layer
        old_coords = []
        for feature in layer:
            sas =
json.loads(feature.ExportToJson())['geometry']['coordinates']
            old_coords.append(sas)

        new_coords = []
        sheet_list = []
        mode_list = []

        #DI - Push the coordinates to the functions that tell their
projection and convert them to the latter projection
        for coord in old_coords:
            tmp_coords = []
            MM,EE,NN,SS = self.calc_everything(coord[0], coord[1])
            if EE != "Null" and NN != "Null" and SS != "Null":
                tmp_coords.append(EE)
                tmp_coords.append(NN)
```

```
                    try:
                        fulu = sheet_list.index(SS)
                    except:
                        sheet_list.append(SS)

                    try:
                        fili = mode_list.index(MM)
                    except:
                        mode_list.append(MM)

                    new_coords.append(tmp_coords)
                    yote_poa = True
                else:
                    yote_poa = False
                    self.iface.messageBar().pushMessage("Error", "One of the
coordinates is outside the 148/1 sheet region of interest!",
level=QgsMessageBar.CRITICAL)

            if yote_poa:
                for sht in sheet_list:
                    self.dlg.lstToposheet.addItem(sht)

                stack_mode = ""
                for mod in mode_list:
                    stack_mode = stack_mode + mod
                self.dlg.lblTransformation.setText(stack_mode)

                if self.dlg.chkLoad.isChecked():
                    oncanvas = True
                else:
                    oncanvas = False

                #DI - return all processed values
                return out_name, new_coords, shp_name, ref, oncanvas
            else:
                self.iface.messageBar().pushMessage("Error", "One of the
coordinates is outside the 148/1 sheet region of interest! Select better
shapefile in Limuru region", level=QgsMessageBar.CRITICAL)
                return "", [], "", "", False

        #DI - Check projection calculation type and send values to transformation
functions
        def calc_everything(self,E,N):
            global mode_calc, new_E, new_N, Sheet
            #for Limuru only
            if E < 0 and N < 0:
                new_E, new_N, Sheet = self.Cas_calc(E,N)
                mode_calc = "Cassini - UTM"

            elif E > 0 and N > 0:
                new_E, new_N, Sheet = self.UTM_calc(E,N)
                mode_calc = "UTM - Cassini"

            return mode_calc, new_E, new_N, Sheet

        #DI - Transform coordinates to UTM
        def Cas_calc(self,E,N):
                global cas_e
                global cas_n
                global sn
                #DI - check if points lie within the subsheet conversion file
                for row in C_S_I:
```

```
                polygon = [(float(row[1]), float(row[2])),(float(row[3]),
float(row[4])),(float(row[5]), float(row[6])),(float(row[7]), float(row[8]))]
                if self.point_in_poly(E,N,polygon):
                    sn = row[0]
                    break
                else:
                    try:
                        pt = (E,N)
                        polygon.index(pt)
                        sn = row[0]
                        break
                    except:
                      sn = "Null"
            #DI - transform
            for row in C_U_P:
                if sn == row[0]:
                    utm_e = (float(row[1]) * E) - (float(row[2]) * N) +
float(row[3])
                    utm_n = (float(row[2]) * E) + (float(row[1]) * N) +
float(row[4])
                    break
                else:
                    utm_e = "Null"
                    utm_n = "Null"

            return utm_e, utm_n, sn

    #DI - Transform coordinates to UTM
    def UTM_calc(self,E,N):
            global cas_e
            global cas_n
            global sn
            #DI - check if points lie within the subsheet conversion file
            for row in U_S_I:
                polygon = [(float(row[1]), float(row[2])),(float(row[3]),
float(row[4])),(float(row[5]), float(row[6])),(float(row[7]), float(row[8]))]
                if self.point_in_poly(E,N,polygon):
                    sn = row[0]
                    break
                else:
                    try:
                        pt = (E,N)
                        polygon.index(pt)
                        sn = row[0]
                        break
                    except:
                      sn = "Null"
            #DI - transform
            for row in U_C_P:
                if sn == row[0]:
                    cas_e = (float(row[1]) * E) - (float(row[2]) * N) +
float(row[3])
                    cas_n = (float(row[2]) * E) + (float(row[1]) * N) +
float(row[4])
                    break
                else:
                    utm_e = "Null"
                    utm_n = "Null"

            return cas_e, cas_n, sn

    #DI - check if points lie in the conversion sheet
    def point_in_poly(self,x,y,poly):
```

```
    n = len(poly)
    inside = False

    p1x,p1y = poly[0]
    for i in range(n+1):
        p2x,p2y = poly[i % n]
        if y > min(p1y,p2y):
            if y <= max(p1y,p2y):
                if x <= max(p1x,p2x):
                    if p1y != p2y:
                        xints = (y-p1y)*(p2x-p1x)/(p2y-p1y)+p1x
                    if p1x == p2x or x <= xints:
                        inside = not inside
        p1x,p1y = p2x,p2y

    return inside
```