Birla Institute of Technology and Science Pilani, Rajasthan

Platform Independent Auto-Registration Tool using GDAL

Practice School I at North Eastern-Space Applications Center

PROJECT REPORT

(13th July 2017)

Prepared in Partial fulfilment of the PS-I Course

Mentors:

Mr. Victor Saikhom
Scientist/Engineer SE

Mr. Siddhartha Bhuyan
Research Scientist

Prepared By:

Akash Manna (2015A3PS225P)

Amit Shukla (2015A3PS238P)

Anjan Das (2015A7PS150P)

Abstract

Image registration is one of the most important steps in most Image Processing tasks for which the final result is achieved from a combination of various resources. Automatic registration of remote-sensing images is a difficult task as it must deal with the intensity changes and variation of scale, rotation and illumination of the images. Our project proposes image registration technique of multi-view, multi- temporal and multispectral remote sensing images. Firstly, a pre-processing step is performed by applying Median Filtering and Histogram Equalization to enhance the images. Secondly, the Steerable Pyramid Transform is adopted to produce multi-resolution levels of reference and sensed images; then, the AKAZE (Accelerated KAZE) algorithm is utilized for extracting feature points that can deal with the large variations of scale, rotation and illumination between images. Thirdly, matching the features points by using the Euclidian distance ratio; then removing the false matching pairs, also known as outliers, using our own version of improved RANdom SAmple Consensus (RANSAC) algorithm. Finally, the mapping function is obtained by the affine transformation using BF (Brute Force) algorithm. The complete application is based on Python language and is based on OpenCV libraries and GDAL libraries as well. For the Graphical User Interface, PyQT4 has been used, since it is an open source software.

Quantitative comparisons of our technique with the related techniques show a significant improvement in the presence of large scale, rotation changes, and the intensity changes. The effectiveness of the proposed technique is demonstrated by the experimental results.

Acknowledgement

We thank our mentors **Victor Saikhom** and **Siddhartha Bhuyan** for their guidance and encouragement in all respects for this project. We also extend our heartfelt gratitude towards our PS instructor **Dr. Kumar Sankar Bhattacharya** for having been there, every-time we needed his help. Last but not the least we sincerely thank **Dr P.L.N Raju**, Director, NESAC and Mr**. Nilay Nishant**, Sci/Engr., NESAC and all the staff members without whose support this project would not have been possible.

We also thank **Prof. Ashoke Kumar Sarkar**, Director of Bits Pilani and **Dr. P. Srinivasan**, Associate Dean, Practice School Division, for giving us this opportunity of getting practical exposure at such a reputed organization.

Table of Contents

1.Organization Profile

1.1. About NESAC

North-Eastern Space Application Centre (NESAC), Umiam established in September, 2000 is a space centre established by the joint initiative of Department of space, Government of India and North Eastern Council. The organization aims to manage the rich natural resources, and improving the poor infrastructure and communication linkages in the North-Eastern region with the help remote sensing based technologies. The motto of NESAC is "Space technology in the service of human kind". The current director of NESAC is Shri P.L.N Raju.

1.2. Facilities at NESAC

1.2.1. RS and GIS Lab: These laboratories provide state of the art computing facility for RS GIS applications. Laboratory is connected to the network of NESAC with 1 Gbps LAN. The lab is equipped with IBM eServer with 4 GB RAM along with more than 25 Workstation based IBM nodes used for digital image processing analysis. The lab also acquired 3 Dual Processor Xeon based DELL Servers with high capacity Storage Area Network (SAN). Recently NESAC has procured high end HP Workstations.

Sufficient numbers of image processing and GIS softwares like Erdas Imagine, Geomatica, ESRI Arc-GIS, e-Cognition etc. along with other software like 3D MAX, Auto-Desk etc. are available in the lab.

Database servers of the lab are also equipped with ORACLE and MS SQL along with Arc-SDE and Arc-IMS capability.

1.2.2. Space And Atmospheric Science Division: The Space Science research group in NESAC is devoted in conducting research to understand the atmospheric processes prevalent over the region through observation and modeling and providing certain services related to agro-meteorology and disaster management. The centre is developing a space science research hub by procuring a series equipments to study the Aerosols, Trace Gases, Solar Radiation, Upper Atmosphere, Atmospheric Boundary Layer, Indian summer monsoon, Thunderstorm, Weather and Climate, etc.

The emphasis is on developing a regional facility to conduct and promote research on Atmospheric and Environmental science in collaboration with the academic institutes and other research organizations in the region.

1.2.3. Sat Com Division: One most important mandate of North Eastern Space Applications Centre (NESAC) is applications of Satellite Communication (SATCOM) technology to undertake developmental communication programmes in the North Eastern Region (NER) to assist education, health, social welfare and other development activities. The SATCOM application programs can addresses the dissemination of quality education- from primary to University level, provisioning of Medical and Health-care services, enabling of interactive connectivity among the rural farming communities , tele-communication support during disaster management etc. To cater this all this ISRO has a number of vibrant SATCOM applications programs like Telemedicine, Te-education, Village Resource Centre, Communication support in disaster management etc. NESAC is playing the key role in implementation and utilization of all above applications programs in all the eight state of NER in collaboration with other central/state government agencies, NGO's. The available SATCOM facilities at NESAC are as follows:

- State-of-the art studio facility for content generation and broadcasting of developmental programs
- Village Resource Centre (VRC) expert node for conducting various training program, providing agro mat advisory and data dissemination
- Satellite Interactive Terminal (SIT) under EDUSAT program for educational activities
- Receive Only Terminal (ROT) under national program on Edusat
- ISRONET system for video conferencing and data transfer activities among the centre of DOS/ISRO
- Transportable WLL-VSAT system for providing audio-video link and data transfer activities for communication support in Disaster management and various training and awareness program
- Various system under Ka-band propagation experiment program
- Various system under GAP-4 experiment program
- Various system under IRNSS project

The Ongoing SATCOM applications programs are as follows:

- Telemedicine
- EDUSAT Utilization Programme
- Village Resource Centre (VRC) program
- Utilization of transportable WLL-VSAT system
- Studio facility at NESAC
- Ka-band propagation experiment
- GSAT-4 Application (GAP-4) Project
- IRNSS Project at NESAC

2. Introduction

For our Practice School (PS-1) program, which spans about eight weeks (22nd May 2017 to 15th July 2017), we were allotted North Eastern Space Applications Center (NESAC), Umiam, as our PS station. Our arrival at NESAC on 22nd of May 2017, was followed by the orientation session in which we were we addressed by Shri P.L.N Raju, Director, NESAC. We were briefed about the working of the institution and its role in the Indian Space Program.

We were introduced to the scientists, some of whom were our mentors. Mr. Nilay Nishant gave us a campus tour and introduced us to the various departments and their ongoing projects.

We were divided into groups of three and randomly allotted to various projects. We were also assigned two mentors under whose guidance we were going to work.

The next day, our mentors, Mr. Victor Saikhom and Mr. Siddhartha Bhuyan, briefed us about our project, i.e., developing a platform-independent Auto-Registration Tool using GDAL. They also told us about their expectations from the project and some suggestive approaches.

The environment at NESAC has its own charm. People here are really hard-working and inspiring. All the staff members were co-operative and friendly. It was a great experience working here.

2. Scope and Limitations

Automatic image registration has been widely studied in the fields of computer vision, remote sensing, and medical imaging. In various cases, such as image fusion, high registration accuracy should be achieved to meet application requirements.

Using manual geo-registration, each set of images (input and reference image) are processed separately under human supervision. This takes a lot of time and efforts as well. Our project addresses this and we aim at creating an auto registration tool which processes an array of set of images, and gives us a set of geo-referenced images. The project can be further applied to geo-reference drone imagery.

However, since the algorithms used are very resource intensive, the program needs at least 8GB of RAM to run. Due to time limitations we were not able to achieve many to one geo-referencing of images, also because of paucity of time we were not able to expand our projection system to support WGS (World Geodetic System) and UTM (Universal Transverse Mercator).

4. About the Project

Image registration is the process of overlaying images of the same scene taken at different times, from different viewpoints, and/or by different sensors. In other words it is a process of finding a transformation that aligns one image to another. The image which is registered is called the reference image and the one which is to be matched to the reference image is called the sensed image.

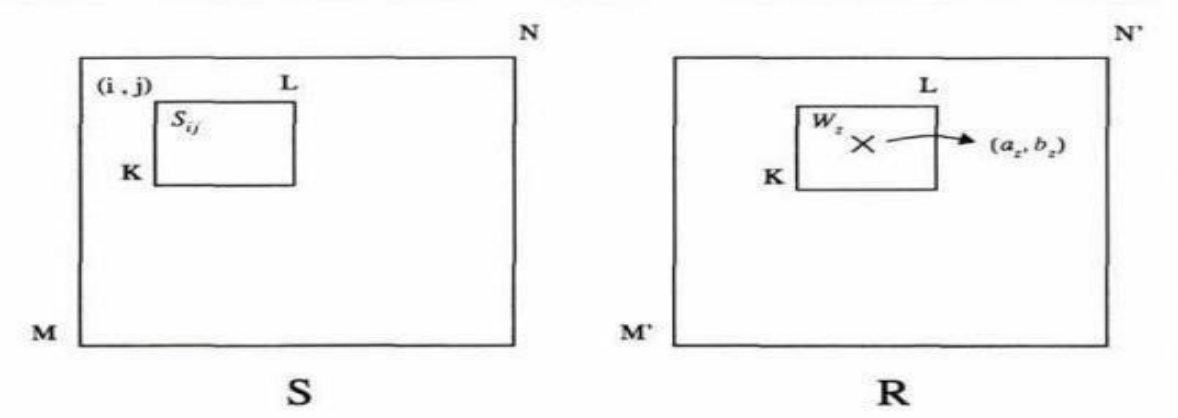
Image registration is the most critical operation in remote sensing applications to enable location based referencing and analysis of earth features. Automatic registration of remote-sensing images is a difficult task as it must deal with the intensity changes and variation of scale, rotation and illumination of the images.

This project aims to perform batch-wise image registration using GDAL. At the same time we use open source software to achieve this[i]. Different algorithms are employed to achieve this, based on pattern recognition, feature based detection, similarity techniques etc. In general, the registration methods are different from each other in the sense that they can combine different techniques for feature identification, feature matching, and mapping functions. The most difficult step in image registration is obtaining the correspondence between the two sets of features. This task is crucial for the accuracy of image registration, and much effort has been spent in the development of efficient feature matching techniques. The traditional manual approach used human assistance to identifying the control points in the images. In this approach, the steps of feature identification and matching are done simultaneously. The images are displayed on the screen and the user chooses corresponding features in the images which clearly appear in both the images. Candidate features include lakes, rivers, coastlines, roads, or other such scene-dominant man-made or natural structures. Each of these features will be assigned one or more point locations (e.g., the centroid of the areas, or line endings, etc.), and these points are referred to as control points. These control points are then used in the determination of the mapping function. In order to get precise registration, a large number of control points must be selected across the whole image. This is a very tedious and repetitive task. Furthermore, this approach requires someone who possesses knowledge of the application domain and is not feasible in cases where there is a large amount of data. Thus, there is a need for automated techniques that require little or no operator supervision. Based on the nature of features used,

automated registration methods can be broadly classified into area-based and feature-based techniques:-

4.1. Area Based Registration

In the area-based methods, a small window of points in the reference image is statistically compared with windows of the same size in the sensed image. The process of Area Based Registration is shown in Figure 1 below.



13.1. Fig. 1 Area based Registration

Area Based Method calculates a certain measurement using gray-data in the fixed-size window from two images and treats the center points of windows as corresponding points when the measured value exceeds the threshold.

The main disadvantage of Area Based Method lies in the excessive computing time spent on searching for the corresponding point.

4.2. Feature Based Registration

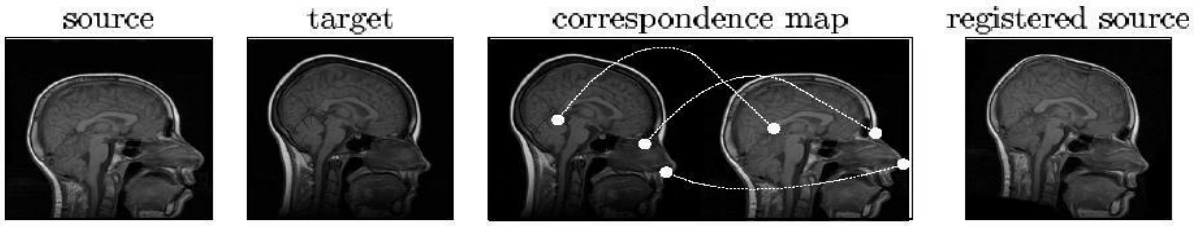
In feature based method algorithms, salient features are extracted from different images, and then corresponding features are determined by comparing the similarities of their descriptions. Matching methods based on point feature, such as SIFT (Scale Invariant Feature Transform) and SURF (Speeded Up Robust Features), are widely used. Other methods based on features, including lines, edges, contours, and shapes, are also used in numerous applications. Figure 2 shown below demonstrates the feature based method of registration.

13.2. Fig. 2 Feature based Registration

The shortcoming of SIFT is excessive memory consumption. Another problem is that for extremely large remotely sensed images, the direct use of SIFT-based matching faces difficulties in obtaining evenly distributed corresponding points.

A basic procedure that we followed is as follows:

Firstly, the Scale Invariant Feature Transform (SIFT) is utilized for extracting feature points that can deal with the large variations of scale, rotation and illumination between images. Secondly, the feature points are matched by using the Euclidean distance ratio, then removing the false matching pairs using the RANSAC (RANdom SAmple Consensus) algorithm.

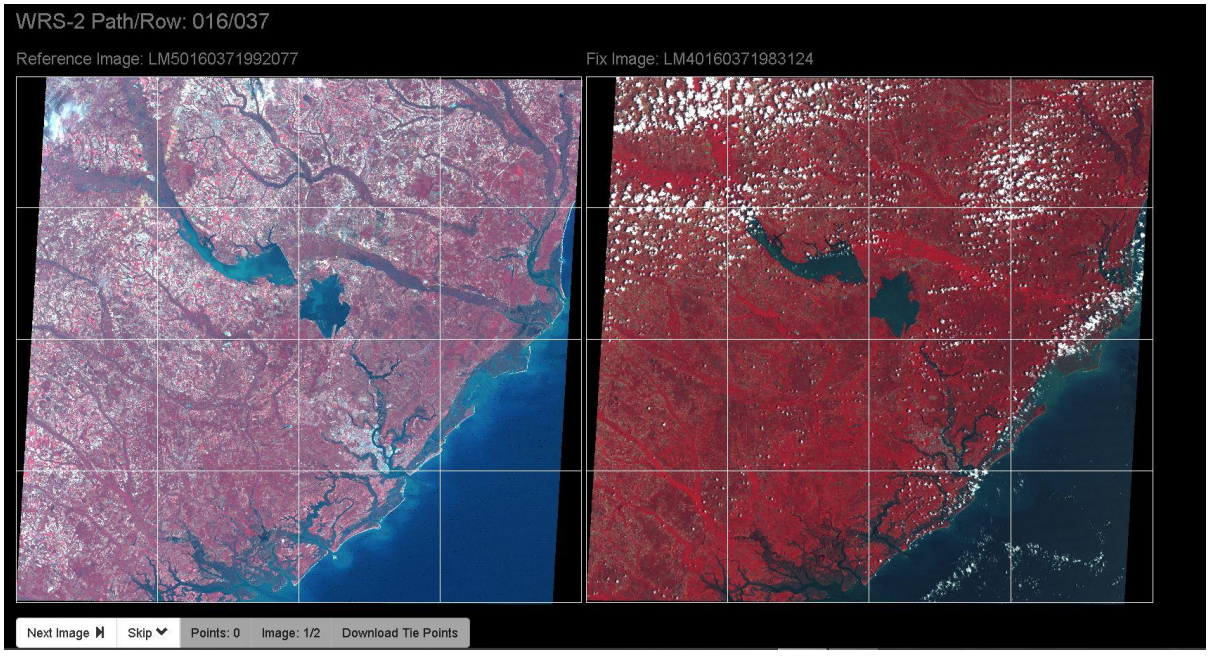


13.3. Fig. 3 Basic Procedure

5. Initial Approach

When we were first introduced to our project, i.e., developing an open-source and platform-independent auto-registration tool, we had very little idea about remote sensing, especially geo-registration. Our mentors gave us a brief orientation session about the background knowledge required to be able to understand the implementation of it and develop the auto-registration tool as well. After getting acquainted with the terminologies and the manual geo-registration procedure, we decided to work in two directions initially, i.e., working in R, RStudio and working in Python.

While working in R, we achieved an interface which could imitate the manual geo-registration procedure quite successfully. The program works using an automated tie-point finding algorithm to find tie points between a given input image and a given reference image. It then uses GDAL to warp the input image based on the tie points that were found. The results were also pretty good. Figure 4 shows the interface which was developed initially in R.



13.4. Fig. 4 R Implementation Interface

On the other hand, in Python also we had done the same already. But, our job was to batch automatize the process for a bulk of image datasets, not just for one. With the current level of knowledge in R language, it was difficult for us to do the task, which is why we picked Python over R.

6. Methodology

Python was chosen as the preferred interpreter, given the availability of scientific and machine learning tools for it. To ensure cross platform GUI, PyQT4 was to be used, which is natively supported by Python. OpenCV will be used for Image Processing Tasks and GDAL (Geospatial Data Abstraction Library) will be used for Registration.

The following Algorithms were tested on actual satellite imagery data (15653x14506x3 and 9473x7983x4), and the results obtained on a system (Intel i7 4710HQ, 8GB RAM, NVIDIA GeForce GTX 960, Windows 10/Ubuntu 16.04) are described:

The project can be divided into the following parts: -

- Image pre-processing
- Find the GCPs by using suitable algorithms using feature detection
- Refine those tie points by removing outliers
- Re-sampling

The Working of the program is as follows: -

6.1. Data Importing

First step is extracting information from the given images. Geospatial raster data is a heavily used product in Geographic Information Systems and Photogrammetry. Input and output images that we are using are represented using raster data. The standard library for loading GIS imagery is the GDAL. We import the GDAL Libraries using the command "**from osgeo import gdal".**

First we open the raster so we can explore its attributes.. GDAL will detect the format if it can, and return a *gdal.Dataset* object.

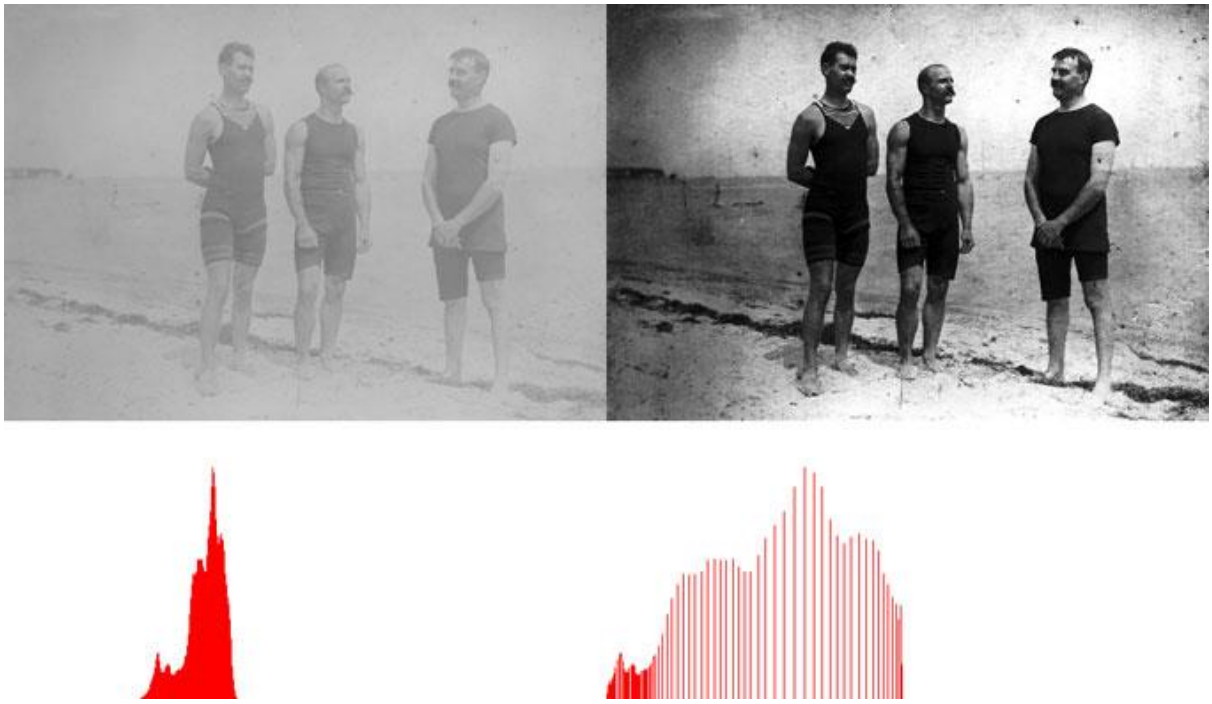
ds = gdal.Open('your_Image.tif')

"array = ds.ReadAsArray()" can be used to read raster data.

6.2. Pre-Processing

After reading the raster data the data is stored as Numpy array as GDAL supports Numpy array. This is followed by normalization of pixel values between 0-255 or an 8 bit size.

The extraction of image data is followed by pre-processing of the image. Image pre-processing is performed using histogram equalization.



Original Image Equalized Image

13.5. Fig. 5 Histogram Equalization

By using histogram equalization you get a much flatter histogram response. This actually increases the overall contrast of the image. Increasing the contrast makes the processing of the images fast and efficient as pixels can be easily differentiated after histogram equalization.

After this image pre-processing step the next is **feature detection.** Feature extraction is done to obtain tie points. A tie point is a point within the image that has the following characteristics:-

- It has a clear, preferably mathematically well-founded, definition,
- It has a well-defined position in image space,
- The local image structure around the interest point is rich in terms of local information contents.

There are basically three aspects of our Methodology, viz., **Feature Detection**, **Descriptor Extraction, Descriptor matching**, **Outlier Rejection** and finally **Image Warping.**

6.3. Feature Detection:

The following algorithms have been used in our program:

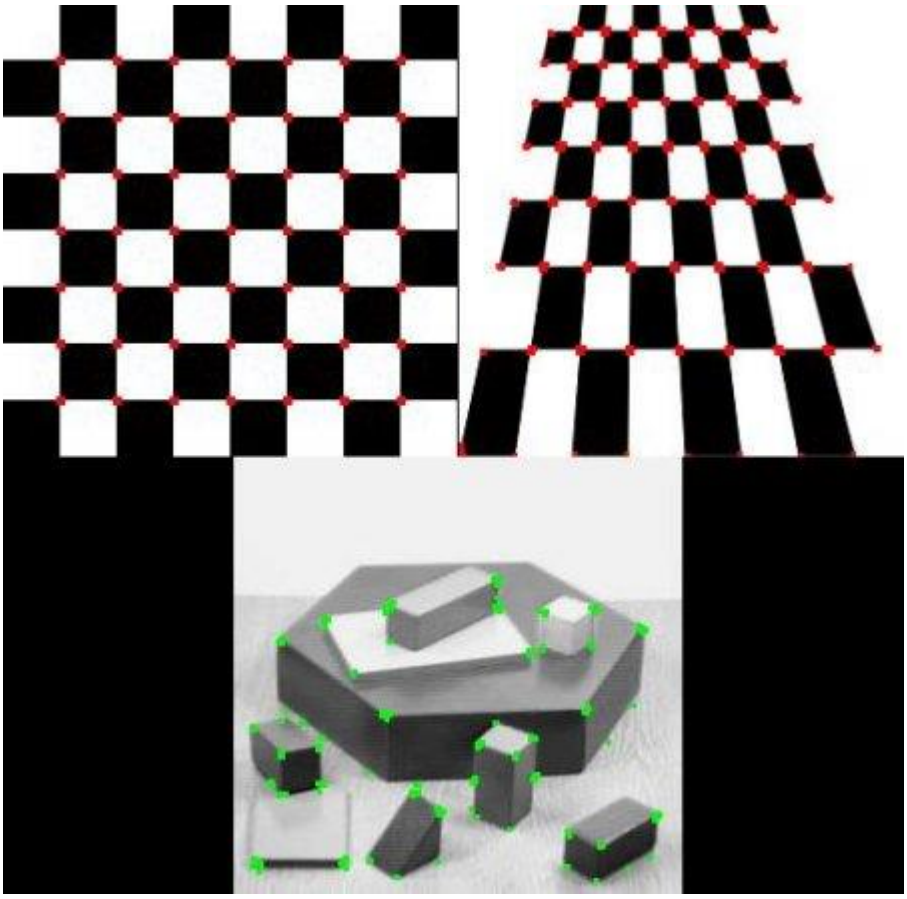
6.3.1. Harris Corner Detection for feature extraction and Cross-Correlation for Point Detection: In this method, the image was split into 5x5 equal sized tiles and the most

prominent corner in each tile was classified as a feature. The corresponding point for each feature on the Reference Image was detected using Cross Correlation with a template image of 50 pixel padding around the feature detected.

6.3.1.1. Outcome: The algorithm is translation and rotation invariant, but is not scale invariant, and is highly susceptible to illumination differences. Also, it requires high processing powers, which are beyond the capacities of normal PC computers as of today.

Time taken to process entire image: 13 minutes

Time taken to process image resized by half: 6 minutes



13.6. Fig. 6 Harris Corner Detection Results

6.3.2. FAST (Features from Accelerated Segment Test) for Feature extraction Correlation with Normed distance: FAST algorithm is based on a paper by Edward Rosten and Tom Drummond - "Machine learning for high-speed corner detection[ii]. For detection, Normed distance was used instead of Euclidean distance and Image was preprocessed by histogram equalization before applying FAST to minimize differences due to illumination variation.

6.3.2.1. Outcome: The algorithm is translation and rotation invariant, and works for images across different illuminations, but is **NOT scale invariant**. It is reasonably fast, and does not require very high processing powers to compute.

Time taken to process entire image: 100-110 sec

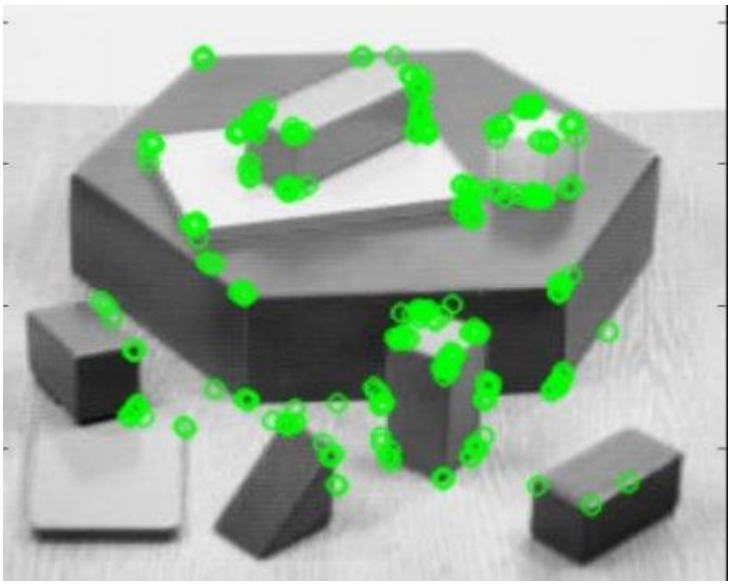
Time taken to process image resized by half: 50 sec

6.3.3. ORB (Oriented-fast and Brisk Rotation): This algorithm uses FAST algorithm for Feature Extraction on both images, and uses BRISK (Binary Robust Invariant Scalable Key points) to find relation between the found key points[iii]. BRISK is a much optimized algorithm. Image was preprocessed by histogram equalization before applying ORB to minimize differences due to Illumination variation.

6.3.3.1. Outcome: The algorithm is translation and rotation invariant, and works for images across different illuminations. The algorithm is **partially scale invariant,** and could handle variations in scale up to a factor of 2. It is very fast, and does not require very high processing powers to compute. This algorithm was found to be the best algorithm to calculate transformation if scale variance is within sufficient bounds.

Time taken to process entire image: 51 sec

Time taken to process image resized by half: 17 sec



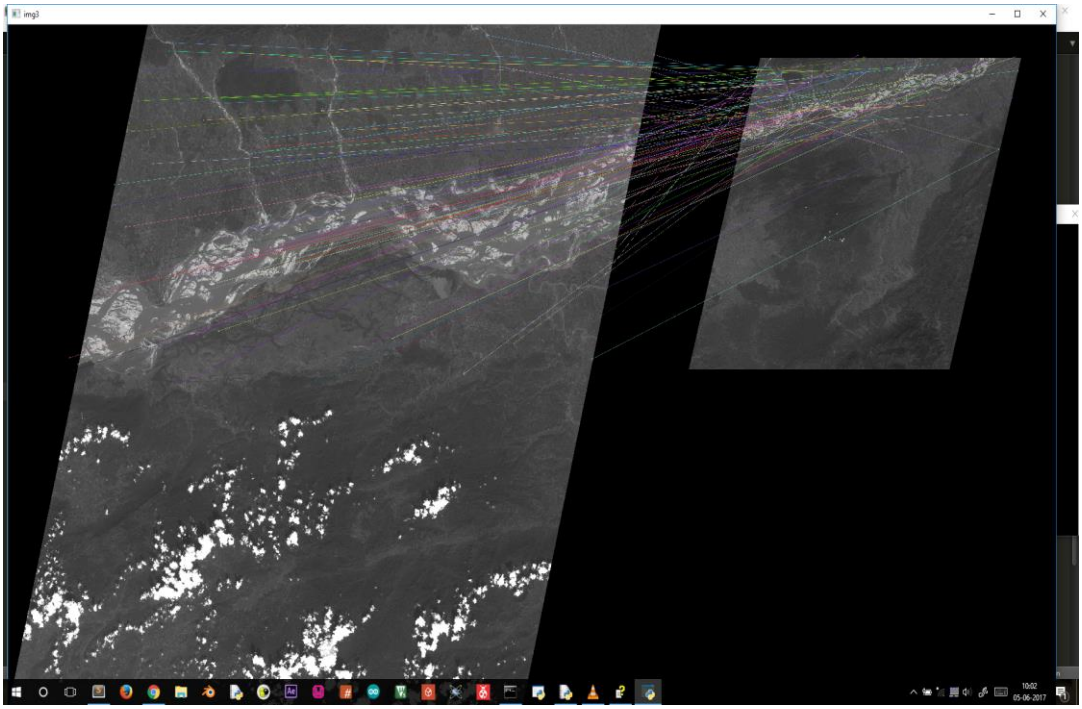
13.7. Fig. 7 ORB Feature Detection Results

6.3.4. SIFT (Scale Invariant Feature Transform) for Feature Extraction and RANSAC (RANdom SAmpling Consensus): This algorithm uses SIFT algorithm for Feature Extraction on BOTH IMAGES, and uses RANSAC to find relation between the found key points[iv]. RANSAC is a training-based machine learning model. The image was pre-processed by histogram equalization before applying ORB to minimize the differences due to illumination variation.

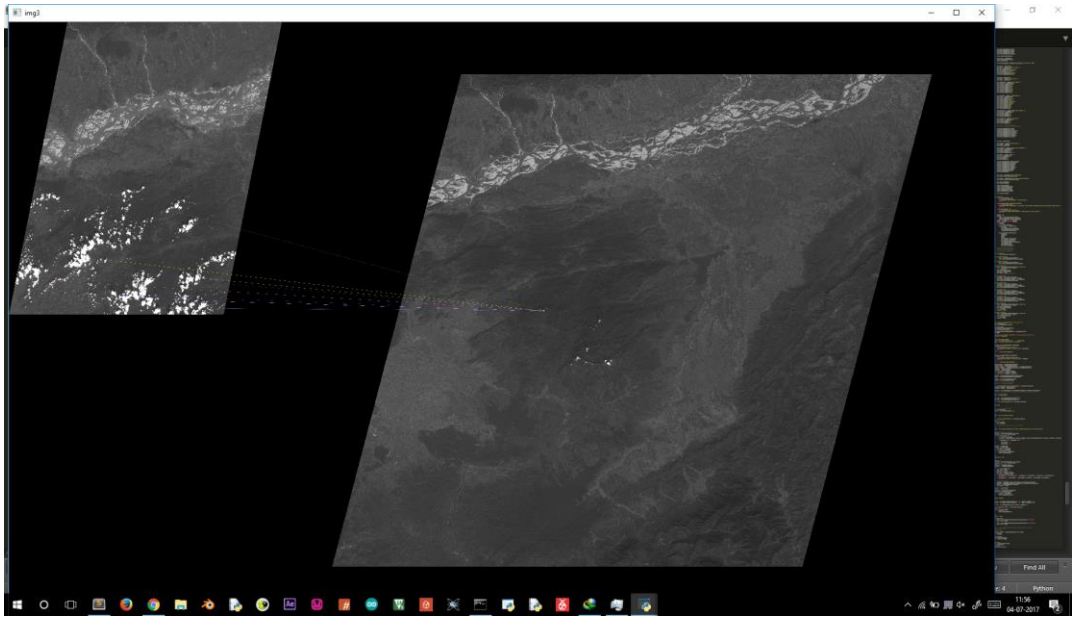
6.3.4.1. Outcome: The algorithm is translation and rotation invariant, and works for images across different illuminations. The algorithm is fully scale invariant, which makes it different from all other algorithms. Calculating SIFT features directly is a very computationally expensive task, but it can be fastened by using SURF (Speeded-Up Robust Features) that reduces the computations to a good extent. This algorithm was found to work robustly even for satellite imagery. Hence, this algorithm was chosen for further work.

Time taken to process entire image (using SURF): 5 minute

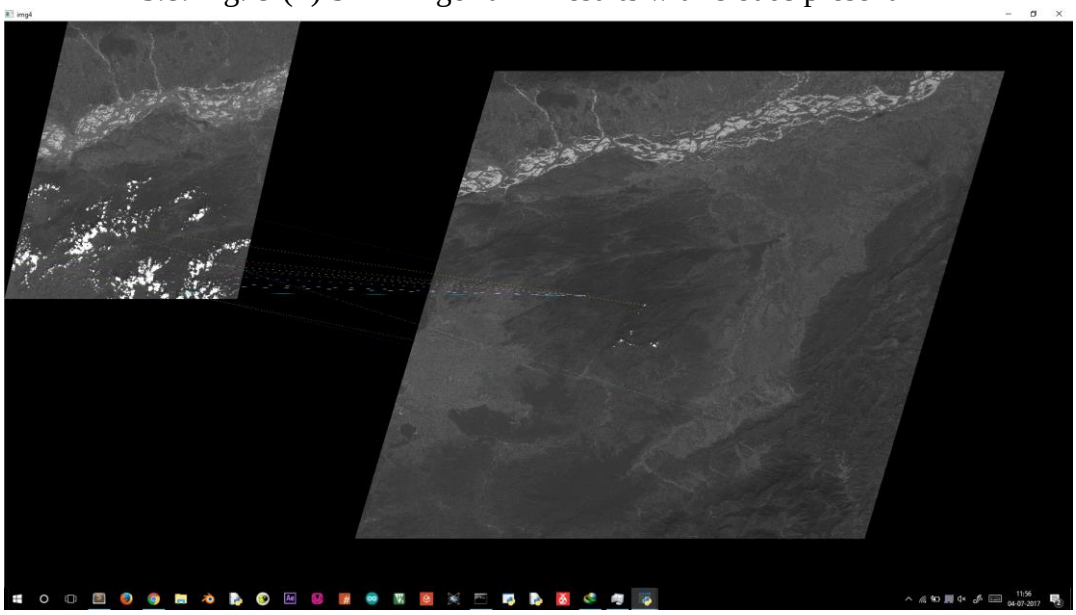
Time taken to process image resized by half: 50 sec



13.8. Fig. 8-(i) SIFT Algorithm Results with no clouds present



13.8. Fig. 8-(ii) SIFT Algorithm Results with clouds present



13.8. Fig. 8-(ii) SIFT Algorithm Results without RANSAC

6.3.5. BRIEF (Binary Robust Independent Elementary Features): BRIEF is a faster method feature descriptor calculation and matching. It also provides high recognition rate unless there is large in-plane rotation

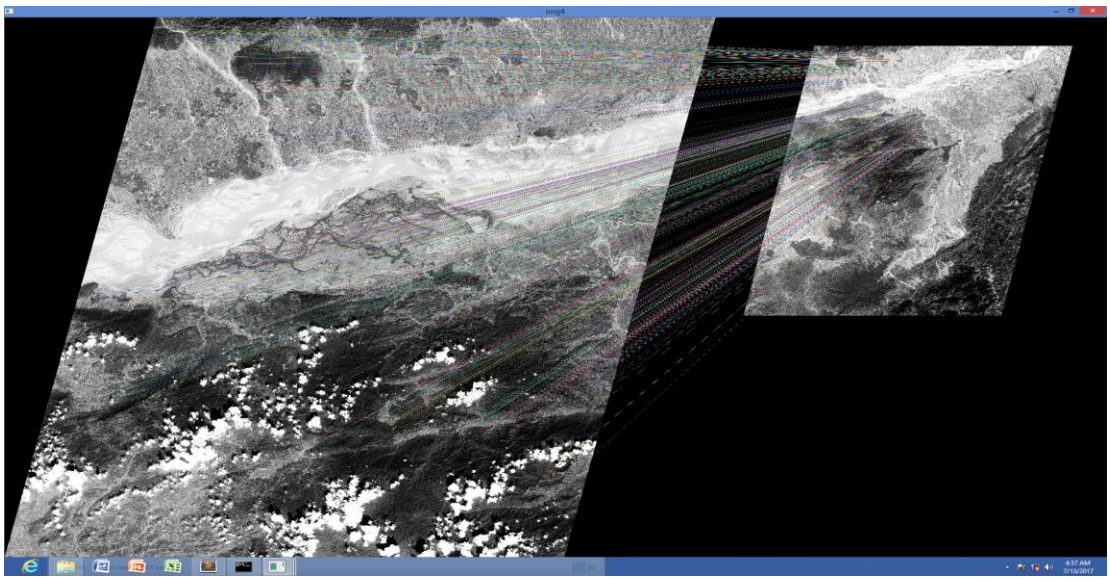
6.3.6. BRISK (Binary Robust Invariant Scalable Keypoints): BRISK relies on an easily configurable circular sampling pattern from which it computes brightness comparisons to form a binary descriptor string. The unique properties of BRISK can be useful for a wide spectrum of applications, in particular for tasks with hard real-time constraints or limited computation power: BRISK finally offers the quality of high-end features in such time-demanding applications. BRISK achieves comparable quality of matching at much less computation time.

13.9 Fig. 9 Feature Matching Using BRISK.

6.3.7. AGAST (Adaptive and Generic Corner Detection Based on the Accelerated Segment Test): This algorithm finds the optimal decision tree in an extended configuration space which can be combined to yield an adaptive and generic accelerated segment test. The resulting method provides high performance for arbitrary environments and so unlike FAST, the corner detector does not have to be trained for a specific scene, but it dynamically adapts to the environment while processing an image.

6.3.8. A-KAZE (Accelerated KAZE): KAZE is an edge-preserving non-linear filtering strategy to locate image features. The filtering is based on the image diffusion equation. A fast explicit diffusion (FED) technique is used to solve the diffusion equation efficiently, contributing to an accelerated variation of the KAZE algorithm, called AKAZE. Keypoints are located by finding the extrema of the second-order derivatives of the image over the non-linear multi-scale pyramid built from the principle of image diffusion.

A-KAZE deploys a technique similar to SURF to estimate the direction of a patch. A modified local difference binary (LDB) representation of the rotation-compensated patch is then extracted as its binary descriptor.

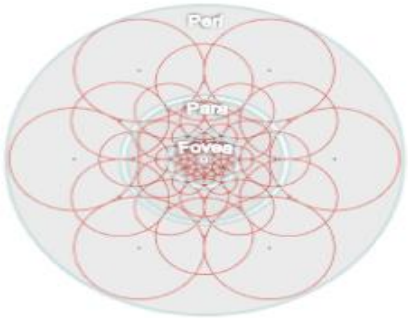


13.10. Fig. 10 AKAZE Feature Detection Results

6.4. Descriptor Extraction

After detecting key points we go on to compute a descriptor for every one of them. A local descriptor is a compact representation of a point's local neighborhood. In contrast to global descriptors, describing a complete object or point cloud, local descriptors try to resemble shape and appearance only in a local neighborhood around a point and thus are very suitable for representing it in terms of matching. **OpenCV** comes with several implementations for feature detection. The algorithms used by us include **SIFT, BRISK, FREAK, AKAZE.**

6.4.1. FREAK (Fast REtinA Keypoint): FREAK suggests using the retinal sampling grid which is also circular with the difference of having higher density of points near the center. The density of points drops exponentially. Each sampling point is smoothed with a gaussian kernel where the radius of the circle illustrates the size of the standard deviation of the kernel.



13.11. Fig. 11 Working of FREAK Algorithm

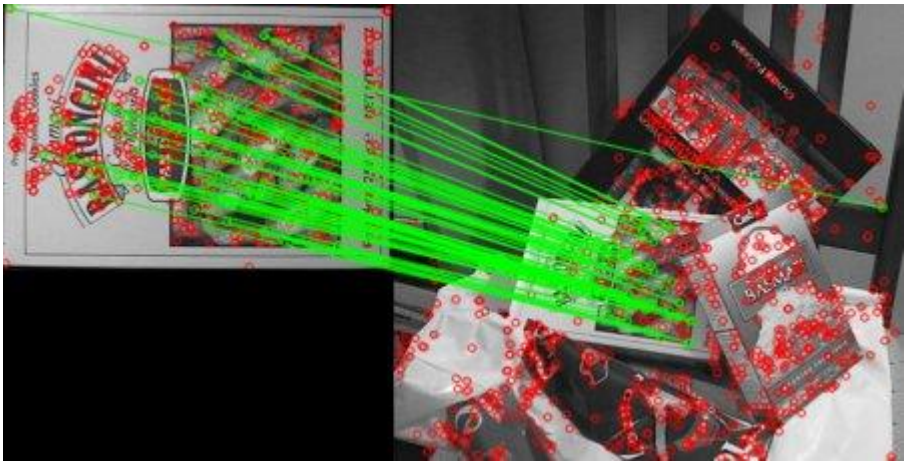
6.5. Correspondence Estimation (Descriptor Matcher)

The next task is to find correspondences between the key points found in both images. Therefore the extracted features are placed in a structure that can be searched efficiently. Usually it is sufficient to look up all local feature-descriptors and match each one of them to its corresponding counterpart from the other image. However due to the fact that two images from a similar scene don't necessarily have the same number of feature-descriptors as one cloud can have more data than the other, we need to run a separate correspondence rejection process. The following algorithms have been taken into use for our program: -

6.5.1. BF (Brute Force): It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

13.12. Fig. 12 Brute Force Algorithm Results

6.5.2. FLANN (Fast Library for Approximate Nearest Neighbors): FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.



13.13. Fig. 13 FLANN Algorithm Results

6.6. Outliers Rejection

The next step is **outliers rejection** or rejecting those points which exceed the threshold of the RMSE (Root Mean Square Error) value. One of the most common approaches to perform correspondence rejection is to use RANSAC (Random Sample Consensus).

6.6.1. RAndom SAmple Consensus (RANSAC): The RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling of observed data. Given a dataset whose data elements contain both inliers and outliers, RANSAC uses the voting scheme to find the optimal fitting result. Data elements in the dataset are used to vote for one or multiple models. The implementation of this voting scheme is based on two assumptions: that the noisy features will not vote consistently for any single model (few outliers) and there are enough features to agree on a good model.

6.7. Image Warping

Changing raster format is done using **gdal_translate.** Suppose you have a raster in UTM coordinates but it is not in .flt format. One can change the format using gdal_translate.

Using **gdalwarp** we can specify output format for the transformation polynomial order and the interpolation.

7. Comparison of Applied Algorithms

	Translation Invariant	Rotation Invariant	Scale Invariant	Illumination Invariant	Process Time* Full Image	Process Time* Half Image
Harris Corner +Correlation	Yes	Partially	No	No	13min	6min
FAST +Correlation*	Yes	Yes	No	Yes	110sec	50sec
ORB + FLANN	Yes	Yes	Partially	Yes	51sec	17sec
SURF + RANSAC	Yes	Yes	Yes	Yes	5min	50sec
AGAST + FREAK	Yes	Yes	Yes	Yes	7min	2min
BRISK	Yes	Yes	Yes	Yes	6.5min	2min
AKAZE	Yes	Yes	Yes	Yes	3min	40sec

22

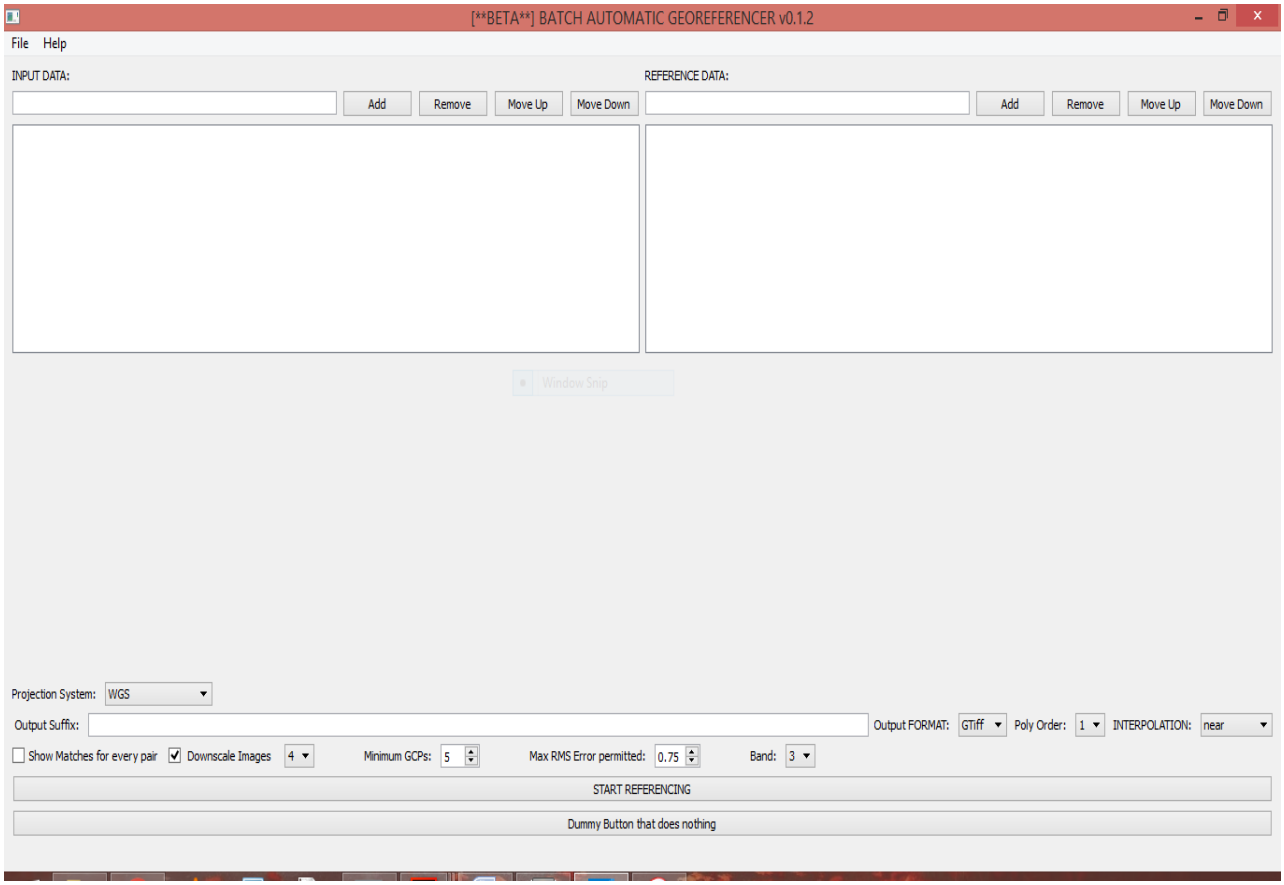
13.14 Fig. 14 Comparative Study of the Algorithms used

8. Development of GUI Using PyQT

The GUI (Graphical User Interface) was developed using PyQT – one of the most famous bindings of Python for the Qt cross-platform. PyQT is an open source free software developed by River Computing. PyQt supports Microsoft Windows as well as various flavors of Unix, including Linux and Mac OS.

PyQt brings together the Qt C++ cross-platform application framework and the cross-platform interpreted language Python. Qt also includes Qt Designer, a graphical user interface designer. PyQt is able to generate Python code from Qt Designer. It is also possible to add new GUI controls written in Python to Qt Designer. PyQt combines all the advantages of Qt and Python. A programmer has all the power of Qt, but is able to exploit it with the simplicity of Python.

Figure 15 shows a snapshot of the program interface which has been developed.



13.15 Fig. 15 Snapshot of the Program Interface

The two main PyQT modules used by in our project are:-

QtCore module: It contains the core non-GUI classes, including the event loop and Qt's signal and slot mechanism.

QtGui module: It contains the majority of the GUI classes.

The functionalities of various buttons and widgets in our GUI are as follows:

Add: To import raster Input and Reference images from the known directory into our main code.

Remove: To remove added images from the list.

Move Up and Move Down: To move an image up or down the list.

Projection System: A drop-down list to select the Projection System of the Input or the Reference Image.

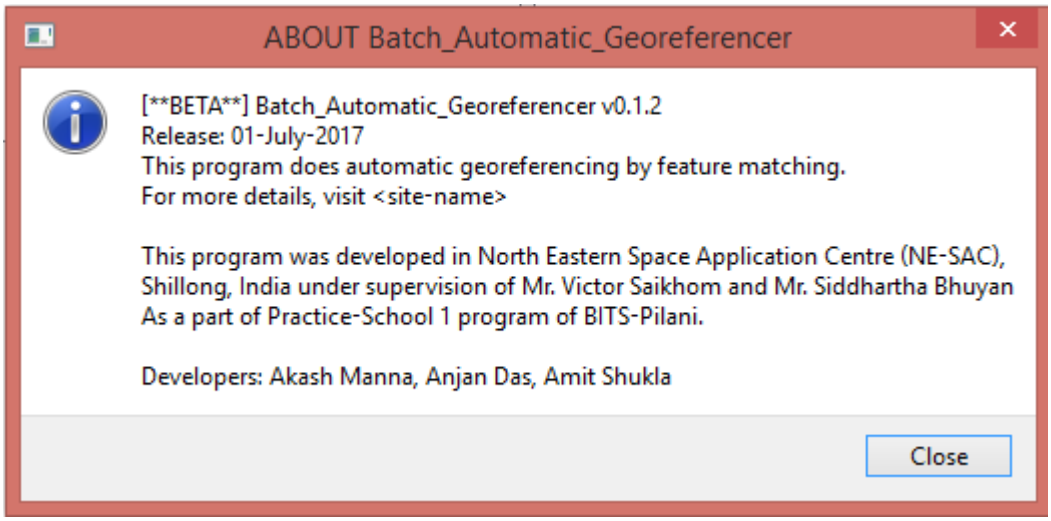
Output Format: To chose the Output Format of the Geo-Referenced Image such as Geo-Tiff etc.

Poly Order: To chose the transformation order 1, 2 or 3

File: Contains the Quit Option. Can also be accessed using the shortcut key Ctrl+Q.

Help: Contains the description about the Project.

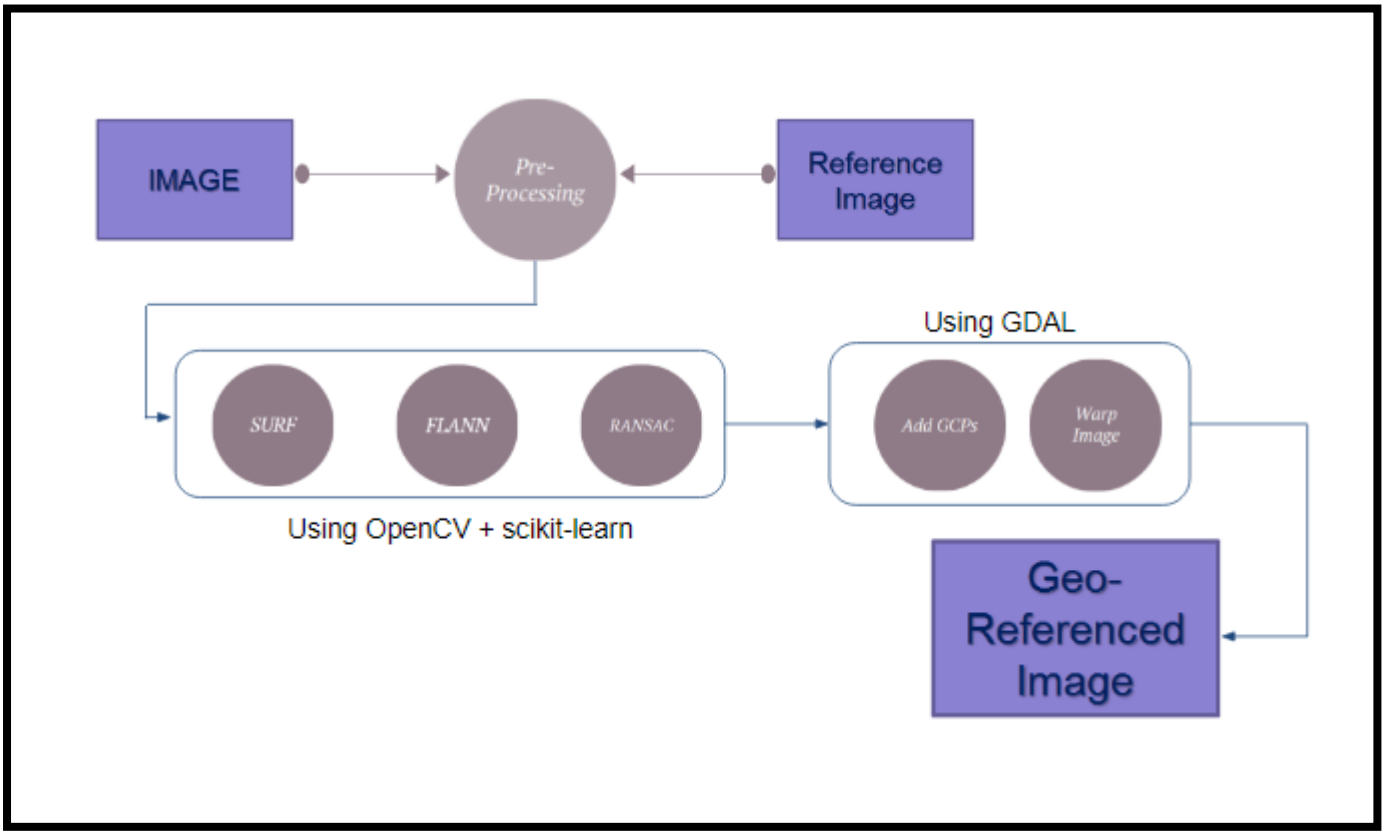
Figure 16 shows the Description window snapshot of the program.



13.16 Fig. 16 Description Window Snapshot

9. FLOWCHART

Figure shows the Image processing procedure which is followed in the background when the program is run.



13.17 Fig. 17 Working Flowchart

10. Results

We were able to geo-reference properly, satellite image data of LISS(Linear Imaging Self Scanner) IV (5.8m x 5.8m) using satellite image data of LISS III (23.6m x 25.9m) with RMS error of less than 0.75 pixels.

Of all the feature extraction algorithms tested, AKAZE detector and descriptor works best, giving maximum correct control points and taking least time. We were able to detect and remove outliers which point to clouds using pattern recognition techniques and cluster detection algorithms. Finally, we were able to do this in a batch, with program automatically detecting reference pairs, thereby minimizing manual interventions further. However, due to unavailability of 3D height information in the satellite images, we could not extend our project to do edge correction.

11. Challenges

- Although the initial model is working for given images, it needs further improvement in terms of efficiency and robustness.
- Presence of clouds was one of the major problems, as clouds cause incorrect tie points to be generated and hence the efficiency of registration reduces.
- Another limitation was regarding high resolution images. Processing of such images consumes too much time.
- Tie points are not being generated always for the scale invariant images with our present model.

12. Future Scope

- Our Application can be further extended to Satellite Drone Imagery.
- The GUI developed can be further improved and can be made more user-friendly.
- The application can be further improved for Parallel Processing using multiple GPUs thereby reducing the processing time significantly.
- There is much scope for use of Ortho-rectification in our application using 3D DEMs (Depth Elevation Models)

12. References

[i] Distinctive Image Features from Scale-Invariant Keypoints David G. Lowe, 5-Jan-2004 https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[ii] Automatic Geo-referencing using Free and Open Source Software Babu, Shekhar et al. http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-8/1121/2014/isprsarchives-XL-8-1121-2014.pdf

[iii] Machine Learning for High Speed Corner Detection http://www.edwardrosten.com/work/rosten_2006_machine.pdf

[iv] Stefan Leutenegger, Margarita Chli and Roland Siegwart: BRISK: Binary Robust Invariant Scalable Keypoints. ICCV 2011: 2548-2555.

13. Appendix

The full Python Script of our application developed, i.e., Batch Automatic Geo-Referencer is listed below.

The first file is named as main.py whose code is as shown:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
import gdal
from gdalconst import *
import time
import sys
import os
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from second import *

inputimgloc = None
refimgloc = None
outimgloc = None
iplist = []
reflist = []

class filedialogdemo(QWidget):
    def __init__(self, parent = None):
        super(filedialogdemo, self).__init__(parent)

        wrapper = QVBoxLayout()

        data = QHBoxLayout()


        inpdata = QVBoxLayout()

        lblInput = QLabel("INPUT DATA:")
        inpdata.addWidget(lblInput)
        loadinpimg = QHBoxLayout()
        self.inpfil = QLineEdit()
        self.inpbtn = QPushButton("+")
        self.inpbtn.clicked.connect(self.getinpfile)
        self.remInp = QPushButton("-")
        self.remInp.clicked.connect(self.removeInpItem)
        self.upInp = QPushButton("^")
        self.upInp.clicked.connect(self.upInpItem)
        self.downInp = QPushButton(".")
        self.downInp.clicked.connect(self.downInpItem)
        loadinpimg.addWidget(self.inpfil)
        loadinpimg.addWidget(self.inpbtn)
        loadinpimg.addWidget(self.remInp)
        loadinpimg.addWidget(self.upInp)
        loadinpimg.addWidget(self.downInp)
        inpdata.addLayout(loadinpimg)

        self.inpList = QListWidget()
        inpdata.addWidget(self.inpList)
        inpdata.addStretch(1)


        refdata = QVBoxLayout()
        lblRef = QLabel("REFERENCE DATA:")
        refdata.addWidget(lblRef)
        loadrefimg = QHBoxLayout()
```

```
58.         #loadrefimg.addStretch(1)
59.         self.reffil = QLineEdit()
60.         self.refbtn = QPushButton("+")
61.         self.refbtn.clicked.connect(self.getreffile)
62.         self.remRef = QPushButton("-")
63.         self.remRef.clicked.connect(self.removeRefItem)
64.         self.upRef = QPushButton("^")
65.         self.upRef.clicked.connect(self.upRefItem)
66.         self.downRef = QPushButton(".")
67.         self.downRef.clicked.connect(self.downRefItem)
68.         loadrefimg.addWidget(self.reffil)
69.         loadrefimg.addWidget(self.refbtn)
70.         loadrefimg.addWidget(self.remRef)
71.         loadrefimg.addWidget(self.upRef)
72.         loadrefimg.addWidget(self.downRef)
73.
74.         refdata.addLayout(loadrefimg)
75.
76.         self.refList = QListWidget()
77.         refdata.addWidget(self.refList)
78.         refdata.addStretch(1)
79.
80.         loadoutimg = QHBoxLayout()
81.         self.outfil = QLineEdit()
82.         self.outbtn = QPushButton("BROWSE")
83.         self.outbtn.clicked.connect(self.getoutfile)
84.         loadoutimg.addWidget(self.outfil)
85.         loadoutimg.addWidget(self.outbtn)
86.
87.         self.runbtn = QPushButton("CHECK MAPPING")
88.         self.runbtn.clicked.connect(self.exex)
89.
90.         self.warpbtn = QPushButton("REFERENCE")
91.         self.warpbtn.clicked.connect(self.runwarp)
92.
93.         data.addLayout(inpdata)
94.         data.addLayout(refdata)
95.
96.         wrapper.addLayout(data)
97.         wrapper.addLayout(loadoutimg)
98.         wrapper.addWidget(self.runbtn)
99.         wrapper.addWidget(self.warpbtn)
100.
101.                 self.setLayout(wrapper)
102.                 self.setWindowTitle("AUTOMATIC GEOREFERENCING - NESAC")
103.                 self.setGeometry(50,100,1000,700)
104.
105.
106.
107.             def exex(self):
108.
109.                 if(self.inpList.count()!=self.refList.count()):
110.                     print("NO WAY SAME COUNT")
111.                     showWarning("COUNT INCONSISTENCY", "The number of Input Images a
    nd Reference Images are different.\nPlease Check.")
112.                     return -1
113.
114.                 inpitems = []
115.                 refitems = []
116.                 for index in range(self.inpList.count()):
117.                     inpitems.append(self.inpList.item(index))
118.                     refitems.append(self.refList.item(index))
119.                 inplocs = [i.text() for i in inpitems]
120.                 reflocs = [i.text() for i in refitems]
121.                 print(inplocs)
122.                 print(reflocs)
123.                 for i in range(len(inplocs)):
124.                     iplist, reflist = run(inplocs[i],reflocs[i])
125.                     gdal_register(self.outfil.text(), inplocs[i], reflocs[i], iplist
    , reflist)
```

```
126.
127.                 #global inputimgloc, refimgloc
128.                 #inputimgloc = self.inpfil.text()
129.                 #refimgloc = self.reffil.text()
130.                 #run(inputimgloc, refimgloc)
131.
132.             def runwarp(self):
133.                 print("y u do this? atleast tell me what i should do ;_;")
134.                 gdal_register(self.outfil.text())
135.                 #DO STUFF
136.
137.             def removeInpItem(self):
138.                 for item in self.inpList.selectedItems():
139.                     self.inpList.takeItem(self.inpList.row(item))
140.
141.             def removeRefItem(self):
142.                 for item in self.refList.selectedItems():
143.                     self.refList.takeItem(self.refList.row(item))
144.
145.             def getinpfile(self):
146.                 fname = QFileDialog.getOpenFileName(self, 'Open file',
147.                  'c:\\',"Image files (*.tif *.img)")
148.                 global inputimgloc
149.                 self.inpfil.setText(fname)
150.                 self.inpList.addItem(fname)
151.                 inputimgloc = fname
152.                 #self.le.setPixmap(QPixmap(fname))
153.
154.             def upInpItem(self):
155.                 currentRow = self.inpList.currentRow()
156.                 currentItem = self.inpList.takeItem(currentRow)
157.                 self.inpList.insertItem(currentRow - 1, currentItem)
158.                 self.inpList.setCurrentItem(currentItem)
159.
160.             def downInpItem(self):
161.                 currentRow = self.inpList.currentRow()
162.                 currentItem = self.inpList.takeItem(currentRow)
163.                 self.inpList.insertItem(currentRow + 1, currentItem)
164.                 self.inpList.setCurrentItem(currentItem)
165.
166.             def upRefItem(self):
167.                 currentRow = self.refList.currentRow()
168.                 currentItem = self.refList.takeItem(currentRow)
169.                 self.refList.insertItem(currentRow - 1, currentItem)
170.                 self.refList.setCurrentItem(currentItem)
171.
172.             def downRefItem(self):
173.                 currentRow = self.refList.currentRow()
174.                 currentItem = self.refList.takeItem(currentRow)
175.                 self.refList.insertItem(currentRow + 1, currentItem)
176.                 self.refList.setCurrentItem(currentItem)
177.
178.             def getreffile(self):
179.                 fname = QFileDialog.getOpenFileName(self, 'Open file',
180.                  'c:\\',"Image files (*.tif *.img)")
181.                 self.reffil.setText(fname)
182.                 self.refList.addItem(fname)
183.                 global refimgloc
184.                 refimgloc = fname
185.
186.             def getoutfile(self):
187.                 fname = QFileDialog.getOpenFileName(self, 'Open file',
188.                  'c:\\',"Image files (*.tif *.img)")
189.                 self.outfil.setText(fname)
190.                 global outimgloc
191.                 outimgloc = fname
192.
193.             def getfiles(self):
194.                 dlg = QFileDialog()
195.                 dlg.setFileMode(QFileDialog.AnyFile)
```

```
196.                 dlg.setFilter("Text files (*.txt)")
197.                 filenames = QStringList()
198.
199.                 if dlg.exec_():
200.                     filenames = dlg.selectedFiles()
201.                     f = open(filenames[0], 'r')
202.
203.                     with f:
204.                         data = f.read()
205.                         self.contents.setText(data)
206.
207.
208.         def showWarning(text="WARNING: ERROR", detailedText=""):
209.                 msg = QMessageBox()
210.                 msg.setIcon(QMessageBox.Critical)
211.
212.                 msg.setText(text)
213.                 msg.setInformativeText(detailedText)
214.                 msg.setWindowTitle("WARNING")
215.                 #msg.setDetailedText(detailedText)
216.                 msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
217.                 msg.exec()
218.
219.         def run(inputimgloc=inputimgloc, refimgloc=refimgloc, rescalefac=4):
220.             time_init = time.time()
221.
222.             '''''
223.             img1 = cv2.imread('clockt.jpg',0)          # queryImage
224.             img2 = cv2.imread('clockt1.jpg',0) # trainImage
225.             '''
226.             inputimg = gdal.Open(inputimgloc, GA_ReadOnly)
227.             if inputimg is None:
228.                 print("FAILED TO IMPORT INPUT IMAGE\n")
229.                 showWarning("INPUT FAILED", "failed to import "+inputimgloc)
230.                 return -1
231.             else:
232.                 print("INPUT IMAGE IMPORTED\n")
233.
234.             #refIMAGE = gdal.Open("112_53_a_22oct12.img", GA_ReadOnly)
235.             refimg = gdal.Open(refimgloc, GA_ReadOnly)
236.             if refimg is None:
237.                 print("FAILED TO IMPORT REFERENCE IMAGE\n")
238.                 showWarning("INPUT FAILED", "failed to import "+refimgloc)
239.                 return -1
240.             else:
241.                 print("REFERENCE IMAGE IMPORTED\n")
242.
243.             inputimg = inputimg.GetRasterBand(1).ReadAsArray()
244.             refimg = refimg.GetRasterBand(1).ReadAsArray()
245.
246.             inputimg = cv2.convertScaleAbs(inputimg)
247.             refimg = cv2.convertScaleAbs(refimg)
248.             #'''
249.
250.             print("LOADING COMPLETE, SIFT CREATION STARTS", time.time()-time_init)
251.
252.             inputimg = cv2.resize(inputimg, (int(15653/rescalefac),int(14506/rescale
    fac)))
253.             refimg = cv2.resize(refimg, (int(9473/rescalefac),int(7983/rescalefac)))

254.
255.             # Initiate SIFT detector
256.             sift = cv2.xfeatures2d.SURF_create()
257.
258.             # find the keypoints and descriptors with SIFT
259.             kp1, des1 = sift.detectAndCompute(inputimg[0:int(8000/rescalefac),:],Non
    e)
260.             kp2, des2 = sift.detectAndCompute(refimg[0:int(3200/rescalefac),0:int(80
    00/rescalefac)],None)
261.
```

```
            print("SIFT DETECTION COMPLETE", time.time()-time_init)

            # BFMatcher with default params
            bf = cv2.BFMatcher()
            matches = bf.knnMatch(des1,des2, k=2)


            print("INITIAL MATCHES FOUND", time.time()-time_init)
            # Apply ratio test
            good = []
            for m,n in matches:
                if m.distance < 0.7*n.distance:
                    good.append([m])

            # cv2.drawMatchesKnn expects list of lists as matches.

            padding = 50
            (ip, ref) = pointsFromMatches(kp1, kp2, good)
            good1 = [] #correlation layer
            for i in range(len(good)):
                (inX, inY) = ip[i]
                (detX, detY) = ref[i]
                inX, inY = int(inX), int(inY)
                detX, detY = int(detX), int(detY)
                print(i, inX, inY, detX, detY)
                if (inX-padding*4<0 or inY-
    padding*4<0 or inX+padding*4>=len(inputimg) or inY+padding*4>=len(inputimg[0])) :
                    continue
                if (detX-padding<0 or detY-
    padding<0 or detX+padding>=len(refimg) or detY+padding>=len(refimg[0])) :
                    continue

                # CORRELATION LAYER
                template2 = refimg[detY-padding:detY+padding, detX-
    padding:detX+padding]
                template1 = inputimg[inY-padding*4:inY+padding*4, inX-
    padding*4:inX+padding*4]
                corrval = correlate(template1, template2)
                if (corrval < 0.1):
                    good1.append(good[i][0])

            src_pts = np.float32([ kp1[m.queryIdx].pt for m in good1 ]).reshape(-
    1,1,2)
            dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good1 ]).reshape(-
    1,1,2)

            M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)

            print("HOMOGRAPHY FOUND", time.time()-time_init)
            good2 = []
            for i in range(len(mask)):
                if(mask[i][0] == 1):
                    good2.append([good1[i]])


            img3 = cv2.drawMatchesKnn(inputimg,kp1,refimg,kp2,good2,None, flags=2)
            drawcv("img3", img3)

            img4 = cv2.drawMatchesKnn(inputimg,kp1,refimg,kp2,[good1],None, flags=2)

            drawcv("img4", img4)


            #img5 = cv2.drawMatchesKnn(inputimg,kp1,refimg,kp2,good,None, flags=2)
            #drawcv("img5", img5)
            global iplist
            global reflist
            (iplist, reflist) = pointsFromMatches(kp1, kp2, good2)
            print(iplist)
            print(reflist)
```

```
325.
326.           #gdal_register(outimgloc, inputimgloc, refimgloc, iplist, reflist)
327.
328.           cv2.waitKey(0)
329.           cv2.destroyAllWindows()
330.           return(iplist, reflist)
331.
332.
333.       def drawGUI():
334.           app = QApplication(sys.argv)
335.           ex = filedialogdemo()
336.           ex.show()
337.           sys.exit(app.exec_())
338.
339.
340.       drawGUI()
```

The second Python script used in our Program is named as second.py whose code is as shown:

```
1.  import numpy as np
2.  import cv2
3.  from matplotlib import pyplot as plt
4.  import gdal
5.  from gdalconst import *
6.  import time
7.  import sys
8.  import os
9.
10.
11. def pixel2coord(ds, x, y):
12.     """Returns global coordinates to pixel center using base-0 raster index"""
13.     '''''
14.     GeoTransform() returns a tuple where (X, Y) are corner coordinates of the image
     indicating the origin of the array,
15.     i.e. data element[0,0]. But, which corner?
16.     If deltaX is positive, X is West.
17.     Otherwise, X is East.
18.     If deltaY is positive, Y is South.
19.     Otherwise, Y is North.
20.     In other words, when both deltaX and deltaY is positive, (X, Y) is the lower-
    left corner of the image.
21.
22.     It is also common to have positive deltaX but negative deltaY which indicates t
    hat (X, Y) is the top-left corner of the image.
23.     There are several standard notations for SRS, e.g. WKT, Proj4, etc.
24.     GetProjection() returns a WKT string.
25.     The meaning and unit-of-
    measurement for  X, Y, deltaX and deltaY are based on the spatial reference system
    (SRS) of the image.
26.     For example, if the SRS is Stereographic projection, they are in kilometres.
27.     '''
28.     xoff, a, b, yoff, d, e = ds.GetGeoTransform()#(X, deltaX, rotation, Y, rotation
    , deltaY) = ds.GetGeoTransform()
29.     xp = a * x + b * y + xoff
30.     yp = d * x + e * y + yoff
31.     return(xp, yp)
32.
33.
34. def gdal_register(outimgloc, inputimgloc, refimgloc, iplist, reflist):
35.     if len(iplist)==0:
36.         print("NO GCPs to translate with.")
37.         return
38.
```

```
    command = "gdal_translate -of GTiff "

    inputimg = gdal.Open(inputimgloc, GA_ReadOnly)
    if inputimg is None:
        print("FAILED TO IMPORT INPUT IMAGE")
    else:
        print("INPUT IMAGE IMPORTED")

    #refIMAGE = gdal.Open("112_53_a_22oct12.img", GA_ReadOnly)
    refimg = gdal.Open(refimgloc, GA_ReadOnly)
    if refimg is None:
        print("FAILED TO IMPORT REFERENCE IMAGE")
    else:
        print("REFERENCE IMAGE IMPORTED")

    for i in range(len(iplist)):
        refx, refy = pixel2coord(refimg, 4*reflist[i][0], 4*reflist[i][1])
        adcmd = "-
gcp " + str(int(4*reflist[i][0])) + " " + str(int(4*reflist[i][1])) + " " + str(ref
x) + " " + str(refy)+ " "
        command = command + adcmd
    command = command + " " + (inputimgloc) + " temp.tif"
    print(command)
    os.system(command)
    print("GCPs added Successfully.")

    print("WARP STARTS")
    cmd2 = "gdalwarp -r near -order 1 -co COMPRESS=NONE temp.tif " + inputimgloc[:-
4] + "output.tif"
    print(cmd2)
    os.system(cmd2)
    os.system("rm -rf temp.tif")
    print("WARP COMPLETE")


def pointsFromMatches(kp1, kp2, matches):
    pairsOfKp1 = [i[0].queryIdx for i in matches]
    pairsOfKp2 = [i[0].trainIdx for i in matches]
    sP = cv2.KeyPoint_convert(kp1, pairsOfKp1)
    dP = cv2.KeyPoint_convert(kp2, pairsOfKp2)
    return sP, dP

def drawcv(imagename, img3):
    screen_res = 1920, 1080
    scale_width = screen_res[0] / img3.shape[1]
    scale_height = screen_res[1] / img3.shape[0]
    scale = min(scale_width, scale_height)
    window_width = int(img3.shape[1] * scale)
    window_height = int(img3.shape[0] * scale)

    cv2.namedWindow(imagename, cv2.WINDOW_NORMAL)
    cv2.resizeWindow(imagename, window_width, window_height)

    cv2.imshow(imagename, img3)

def draw2cv(imagename, imgA, imgB):
    imgAx = len(imgA[0])
    imgAy = len(imgA)
    imgBx = len(imgB[0])
    imgBy = len(imgB)

    imgA = cv2.resize(imgA, (imgBx, imgBy))
    img = np.hstack((imgA, imgB))
            drawcv(imagename, img)
            #plt.imshow(img)
            #plt.show()

       def correlate(img, template):
            w, h = template.shape[::-1]
```

```
106.            # All the 6 methods for comparison in a list
107.            #methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
108.            #            'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORM
    ED']
109.            #for meth in methods:
110.            meth = 'cv2.TM_SQDIFF_NORMED'
111.            method = eval(meth)
112.
113.            # Apply template Matching
114.            res = cv2.matchTemplate(img,template,method)
115.            min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
116.            #print("correlate called")
117.            # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
118.            if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
119.                return min_val
120.            else:
121.                return max_val
122.            #bottom_right = (top_left[0] + w, top_left[1] + h)
123.            #cv2.rectangle(img,top_left, bottom_right, 255, 2)
124.            #drawcv(img)
```