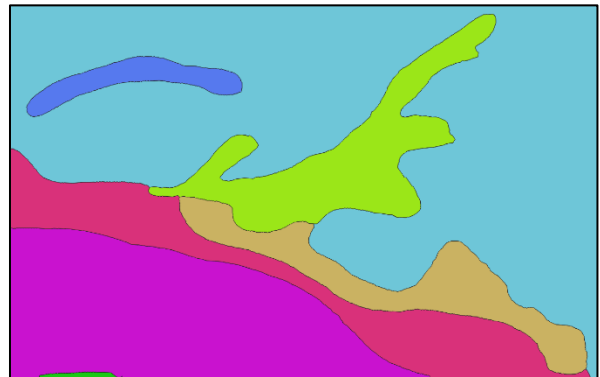
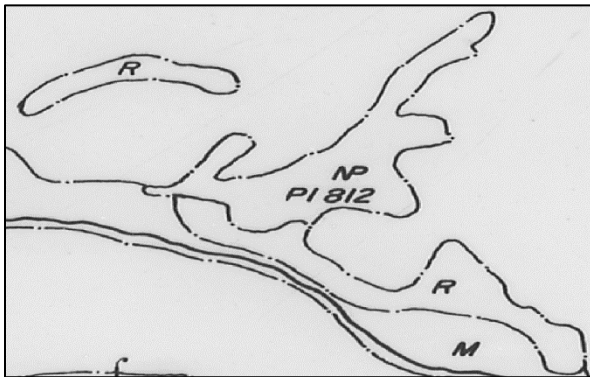


# User Guide

## Forest Cover Map Segmentation Tool

Team Sigma Brains (Joseph Chang, Christopher Hernandez, and Jung-han Han)  
Camosun College ICS Capstone 2022

Recipient: NFIS team at the Pacific Forestry Centre, Natural Resources Canada  
Submission Date: 08-11-2022



## Table of Contents

Introduction .....	1
Instructions .....	2
System Requirements .....	2
Installation .....	2
Python development libraries installation.....	2
GDAL development libraries installation .....	2
Cloning the FMS tool git repository into the host .....	3
Python virtual environment creation.....	3
Python virtual environment activation .....	3
Python package dependencies installation .....	3
File Structure .....	4
Running the FMS Tool.....	6
Python virtual environment activation .....	6
Running the FMS tool to convert GeoTiff files to Shapefile .....	6
Result files .....	8
Conclusion.....	10
Appendix .....	11
Appendix A: Developers' Manual .....	A-1
The Algorithm of the FMS tool.....	A-1
Dot-dashed line extraction .....	A-2
Polygonizing the extracted lines .....	A-3
Label extraction.....	A-4
Parameters of the algorithm.....	A-5
Major issues needed to be resolved .....	A-7
Finding the optimal parameter values.....	A-7
Dot-dashed line merging into solid line .....	A-7
Symbols overlapping with dot-dashed lines .....	A-8

## List of Figures

Figure 1. Virtual environment 'fms-env' activated .....	3
Figure 2. Python package dependencies successfully installed.....	3
Figure 3. File Structure of the FMS tool .....	4
Figure 4. '093C10f_1975_D_1_clipped_small.tif' to Shapefile.....	6
Figure 5. Images processed in parallel.....	7
Figure 6. GeoTiff file named '093C10f_1975_D_1_clipped_small.gtiff' .....	8
Figure 7. The Shapefile of the extracted dot-dashed lines .....	8
Figure 8. The Shapefile of the extracted polygons .....	8
Figure 9. The extracted label information associated with polygons in the Shapefile.....	9
Figure 10. The content of an example log file '093C10f_1975_D_1_clipped_small.txt' .....	9
Figure 11. Main flow of the algorithm .....	A-1
Figure 12. Flow of the line extraction algorithm .....	A-2
Figure 13. An example of training sample .....	A-4
Figure 14. Dot-dashed line merging into solid line .....	A-7
Figure 15. Overlapping symbols.....	A-8

## Introduction

The Forest Cover Map Segmentation (FMS) Tool is a tool which converts historical, hand-drawn, scanned, forest cover map files into geospatial vector data. The purpose of this program is to detect sections of a GeoTiff file, extract the data, and present it in an analysis-ready file format called Shapefile. In addition, the algorithm will take sections of the map which are bordered with hand-drawn dots and dashes and draw solid lines which will form enclosed polygon shapes to represent the sections. Using character recognition, the program also extracts label data from each region polygon and associates the extracted label information with the corresponding region.

## Instructions

### System Requirements

Although the FMS tool can run on both Windows 10 and Linux, it is recommended to run it on Linux. This is because several Python packages used in the FMS tool are not officially supported on Windows 10.

In terms of hardware requirement, since the FMS tool processes large, scanned images, it is recommended to have enough RAM. The required RAM size may differ depending on the size of input images.

These are the detailed system requirements:

- Operating System (OS)
  - Linux (Tested on Ubuntu Server 18.04)
- RAM
  - 32GB+
- Python Version
  - 3.6+

### Installation

The following instruction assumes that Ubuntu is used. Each step demonstrates the commands that need to be run.

#### Python development libraries installation

Run the following commands sequentially.

```
sudo apt-get update
sudo apt-get install python3-venv
sudo apt-get install python3-dev
sudo apt-get install python3-opencv
```

#### GDAL development libraries installation<sup>1</sup>

Run the following commands sequentially.

```
sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get update
sudo apt-get install gdal-bin
sudo apt-get install libgdal-dev
export CPL_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
```

---

<sup>1</sup> Installed versions for test: 2.4.2

Cloning the FMS tool git repository into the host  
Run the following command.

```
git clone https://github.com/sigmaprains/forest-map-segmentation.git
```

Python virtual environment creation

Run the following commands sequentially.

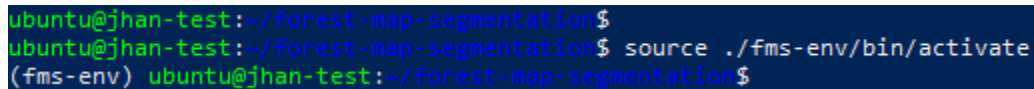
```
cd forest-map-segmentation
python -m venv fms-env
```

*You may need to type `python3` instead of `python` if your system has `python` command linked to Python 2.*

Python virtual environment activation

Run the following command. If the virtual environment `fms-env` is activated, the terminal prompt will indicate that at its front as shown in Figure 1.

```
source ./fms-env/bin/activate
```



```
ubuntu@jhan-test:~/forest-map-segmentation$
ubuntu@jhan-test:~/forest-map-segmentation$ source ./fms-env/bin/activate
(fms-env) ubuntu@jhan-test:~/forest-map-segmentation$
```

Figure 1. Virtual environment 'fms-env' activated

Python package dependencies installation

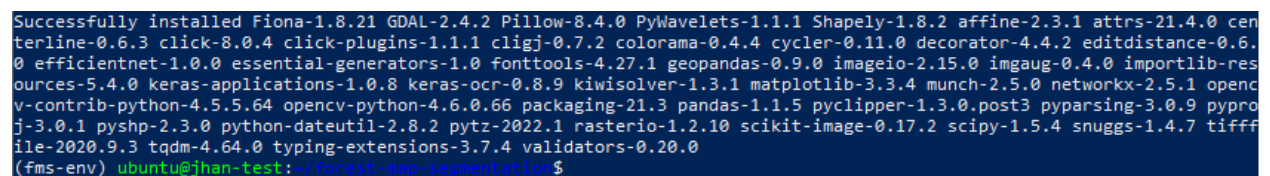
Run the following commands sequentially.

If the Python package dependencies are successfully installed, the installed packages will be displayed as shown in Figure 2.

```
pip install setuptools==57.5.0
pip install --upgrade pip
pip install -r requirements.txt
```

*If you get `ReadTimeoutError` while installing `Tensorflow`, try `pip install tensorflow==2.6.2 --default-timeout=1000` command to install `Tensorflow` independently.*

`requirements.txt` contains a list of the required python packages and their versions for the FMS tool.



```
Successfully installed Fiona-1.8.21 GDAL-2.4.2 Pillow-8.4.0 PyWavelets-1.1.1 Shapely-1.8.2 affine-2.3.1 attrs-21.4.0 cen
terline-0.6.3 click-8.0.4 click-plugins-1.1.1 cligj-0.7.2 colorama-0.4.4 cycloper-0.11.0 decorator-4.4.2 editdistance-0.6.
0 efficientnet-1.0.0 essential-generators-1.0 fonttools-4.27.1 geopandas-0.9.0 imageio-2.15.0 imgaug-0.4.0 importlib-res
ources-5.4.0 keras-ocr-0.8.9 kiwisolver-1.3.1 matplotlib-3.3.4 munch-2.5.0 networkx-2.5.1 openc
v-contrib-python-4.5.5.64 opencv-python-4.6.0.66 packaging-21.3 pandas-1.1.5 pyclicker-1.3.0.post3 pyparsing-3.0.9 pypro
j-3.0.1 pyshp-2.3.0 python-dateutil-2.8.2 pytz-2022.1 rasterio-1.2.10 scikit-image-0.17.2 scipy-1.5.4 snuggs-1.4.7 tifff
ile-2020.9.3 tqdm-4.64.0 typing-extensions-3.7.4 validators-0.20.0
(fms-env) ubuntu@jhan-test: ~/forest-map-segmentation$
```

Figure 2. Python package dependencies successfully installed

## File Structure

The directory `forest-map-segmentation` contains files and directories relevant to the FMS tool. The structure of this directory is as depicted in Figure 3.

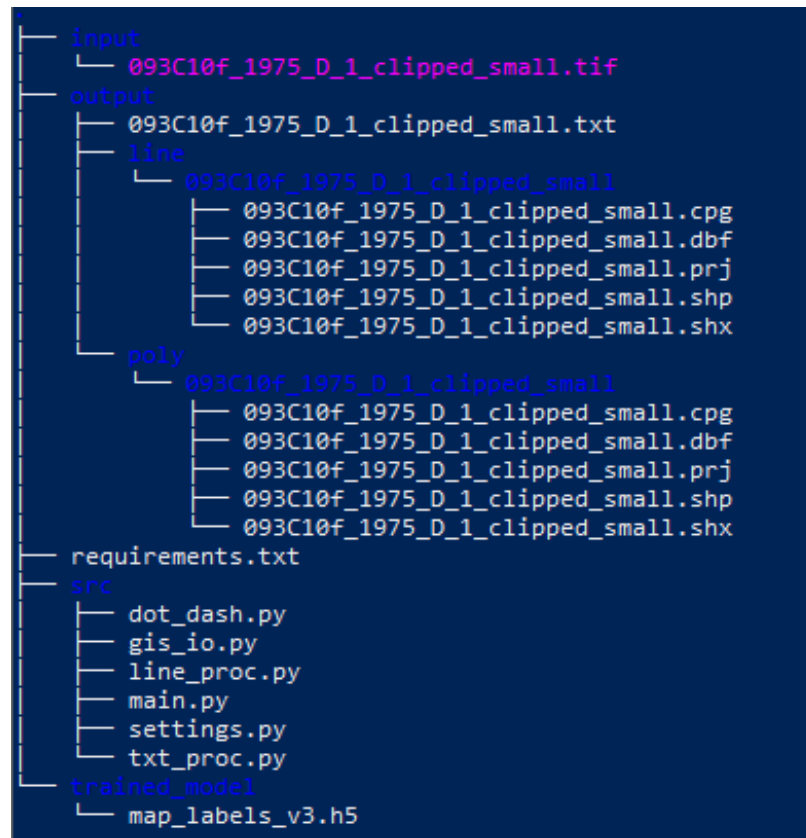


Figure 3. File Structure of the FMS tool

- input**  
 This directory contains input GeoTiff files. However, this directory is just to help organize input files. The user can put input images in any directories other than the `input` directory.
- output**  
 This directory contains log files (e.g., `093C10f_1975_D_1_clipped_small.txt`) for each processed image. In addition, this directory contains the `line` and the `poly` directories which contain output Shapefiles. The `line` directory contains the extracted lines as Shapefiles. The `poly` directory contains the extracted polygons and their corresponding label information as Shapefiles.  
*The Shapefiles in the `poly` directory are the final output of the FMS tool.*
- src**  
 This directory contains Python script files of the FMS tool. The user should execute `main.py` to run the FMS tool. `settings.py` defines all parameters that control how the FMS tool processes images.

- `trained_model`

This directory contains a file (i.e., `map_labels_v3.h5`) that stores the weights and model configuration of our Keras-OCR AI model trained with our own dataset.



## Running the FMS Tool

Python virtual environment activation

Run the following command.

```
source ./fms-env/bin/activate
```

*If the virtual environment is already activated in the process of installation, you can skip this step.*

Running the FMS tool to convert GeoTiff files to Shapefile

*Converting a single GeoTiff file*

```
usage: main.py input_gtiff_file
```

The following command will convert a GeoTiff file named '093C10f\_1975\_D\_1\_clipped\_small.tif' to Shapefile. If the conversion is successfully done, the output in the terminal will look like Figure 4.

```
python src/main.py input/093C10f_1975_D_1_clipped_small.tif
```

*You can ignore the message displaying 'Provided alphabet does not match pretrained alphabet.' since our program uses the AI model trained with our own dataset.*

```
(fms-env) ubuntu@jhan-test:~/forest-map-segmentation$ python src/main.py input/093C10f_1975_D_1_clipped_small.tif
1:Extracting dot-dashed lines
1:Polygonizing the extracted dot-dashed lines
1:Extracting labels
Provided alphabet does not match pretrained alphabet. Using backbone weights only.
Looking for /home/ubuntu/.keras-ocr/crnn_kurapan_notop.h5
Looking for /home/ubuntu/.keras-ocr/craft_mlt_25k.h5
1:Writing Shapefile
1:Done Conversion
All output files are stored in: /home/ubuntu/forest-map-segmentation/output
(fms-env) ubuntu@jhan-test:~/forest-map-segmentation$
```

Figure 4. '093C10f\_1975\_D\_1\_clipped\_small.tif' to Shapefile.

### *Converting multiple GeoTiff files in parallel*

```
usage: main.py input_gtiff_file1 input_gtiff_file2 input_gtiff_file3 ...
```

You can process multiple GeoTiff files at the same time. They will be processed in parallel. The following command will convert three GeoTiff files named 'case1.tif', 'case2.tif', and 'case3.tif'. If the conversion is successfully done, the output in the terminal will look like Figure 5. The maximum number of images that can be processed in parallel depends on the number of CPU cores of the system.

```
python src/main.py input/case1.tif input/case2.tif input/case3.tif
```

*The number at the beginning of each output message indicates the index of the process.*

```
(fms-env) ubuntu@jhan-test:~/forest-map-segmentation$ python src/main.py input/case1.tif input/case2.tif input/case3.tif
1:Extracting dot-dashed lines
2:Extracting dot-dashed lines
3:Extracting dot-dashed lines
1:Polygonizing the extracted dot-dashed lines
1:Extracting labels
2:Polygonizing the extracted dot-dashed lines
2:Extracting labels
3:Polygonizing the extracted dot-dashed lines
3:Extracting labels
Provided alphabet does not match pretrained alphabet. Using backbone weights only.
Looking for /home/ubuntu/.keras-ocr/crnn_kurapan_notop.h5
Looking for /home/ubuntu/.keras-ocr/craft_mlt_25k.h5
Provided alphabet does not match pretrained alphabet. Using backbone weights only.
Looking for /home/ubuntu/.keras-ocr/crnn_kurapan_notop.h5
Looking for /home/ubuntu/.keras-ocr/craft_mlt_25k.h5
Provided alphabet does not match pretrained alphabet. Using backbone weights only.
Looking for /home/ubuntu/.keras-ocr/crnn_kurapan_notop.h5
Looking for /home/ubuntu/.keras-ocr/craft_mlt_25k.h5
1:Writing Shapefile
3:Writing Shapefile
1:Done Conversion
3:Done Conversion
2:Writing Shapefile
2:Done Conversion
All output files are stored in: /home/ubuntu/forest-map-segmentation/output
(fms-env) ubuntu@jhan-test:~/forest-map-segmentation$
```

*Figure 5. Images processed in parallel*

## Result files

When the conversion from a GeoTiff file to Shapefile is done, the output files are stored in the **output** directory. Here, we show the result files obtained after the FMS tool processed a GeoTiff file named '093C10f\_1975\_D\_1\_clipped\_small.gtiff'. Figure 6 shows this GeoTiff file.

There are three types of output files created when a GeoTiff file is processed:

- **Line extraction result** – The extracted dot-dashed lines are stored as Shapefiles in **line** directory in **output** directory. This directory contains the extracted lines as Shapefiles. Figure 7 depicts the extracted dot-dashed lines opened in a Geographic Information System (GIS) application named QGIS<sup>2</sup>.
- **Polygon and label extraction result** – The extracted polygons and their corresponding label information are stored as Shapefiles in **poly** directory in **output** directory. This is the final output of the FMS tool. Figure 8 shows the extracted polygons, and Figure 9 depicts the extracted labels opened in QGIS.
- **Log file** – It is a text file that contains information on how the image file was processed. If there were errors while the FMS tool was processing a GeoTiff file, the detailed error messages would be recorded in this log file. Figure 10 shows the content of the log file created after processing the GeoTiff file. The log file's name is the same as the processed GeoTiff file name except for its extension (i.e., .txt).

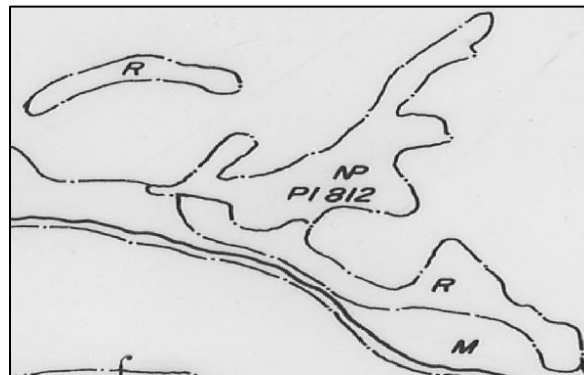


Figure 6. GeoTiff file named '093C10f\_1975\_D\_1\_clipped\_small.gtiff'



Figure 7. The Shapefile of the extracted dot-dashed lines

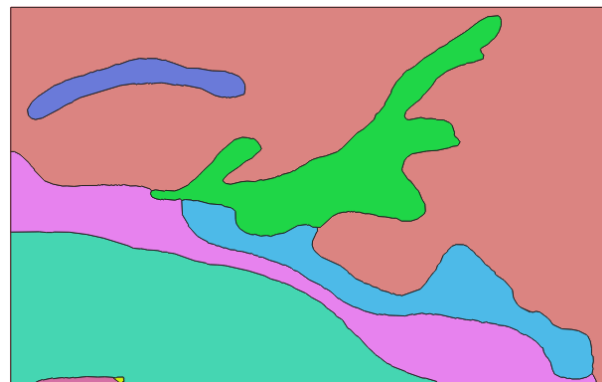


Figure 8. The Shapefile of the extracted polygons

<sup>2</sup> QGIS is a free and open-source cross-platform desktop GIS application. There are many other GIS applications such as ArcGIS and GeoMedia that can read Shapefile.

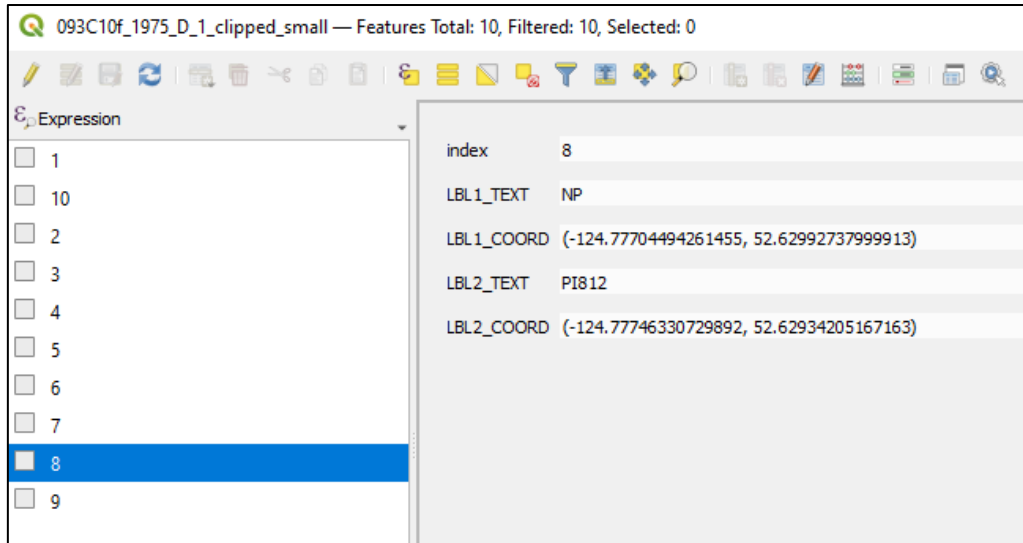


Figure 9. The extracted label information associated with polygons in the Shapefile

```

2022-07-30 23:26:47,341 | root | INFO | Detecting dots
2022-07-30 23:26:47,394 | root | INFO | Getting GeoJSON from image
2022-07-30 23:26:47,418 | root | INFO | Number of found polygons : 92
2022-07-30 23:26:47,418 | root | INFO | Getting Shapely polygons from GeoJSONs
2022-07-30 23:26:47,421 | root | INFO | Extracting dot dashed lines
2022-07-30 23:26:47,433 | root | INFO | Searching dash polygons around the detected dots
2022-07-30 23:26:51,465 | root | INFO | Making virtual dots and searching dash polygons around them
2022-07-30 23:26:51,994 | root | INFO | Extracting the lines from dots and dash polygons
2022-07-30 23:26:51,995 | root | INFO | Finding shortest path between dots to connect dots and dashes
2022-07-30 23:26:51,996 | root | INFO | Filtering out long lines
2022-07-30 23:26:51,996 | root | INFO | Adding extracted dot dash lines
2022-07-30 23:26:51,997 | root | INFO | Finding shortest path between dots to connect dots and dashes
2022-07-30 23:26:51,997 | root | INFO | Filtering out long lines
2022-07-30 23:26:51,997 | root | INFO | Adding extracted dot dash lines
2022-07-30 23:26:51,998 | root | INFO | Finding shortest path between dots to connect dots and dashes
2022-07-30 23:26:51,998 | root | INFO | Filtering out long lines
2022-07-30 23:26:51,998 | root | INFO | Adding extracted dot dash lines
2022-07-30 23:26:51,999 | root | INFO | Finding shortest path between dots to connect dots and dashes
2022-07-30 23:26:52,000 | root | INFO | Filtering out long lines
2022-07-30 23:26:52,000 | root | INFO | Adding extracted dot dash lines
2022-07-30 23:26:52,000 | root | INFO | Finding shortest path between dots to connect dots and dashes
2022-07-30 23:26:52,001 | root | INFO | Filtering out long lines
2022-07-30 23:26:52,001 | root | INFO | Adding extracted dot dash lines
2022-07-30 23:26:52,001 | root | INFO | Finding shortest path between dots to connect dots and dashes
2022-07-30 23:26:52,001 | root | INFO | Filtering out long lines

```

Figure 10. The content of an example log file '093C10f\_1975\_D\_1\_clipped\_small.txt'

## Conclusion

You can now use the FMS Tool at its fullest and create meaningful data. We hoped to make an easy-to-read instruction manual for all application users who will use our application. If you need assistance, you can check out the contact information below.

In case of difficulties when using the application, you can contact a supervisor of the project with the following link:

[https://ca.nfis.org/contact\\_eng.html](https://ca.nfis.org/contact_eng.html)

# Appendix

## Appendix A: Developers' Manual

In this appendix, we explain the things developers need to know when they experiment and modify the program. We explain how the algorithm of the program works, the parameters for tuning the program, and the detailed explanation on major issues that need to be fixed.

### The Algorithm of the FMS tool

The main flow of the algorithm is as shown in Figure 11. We explain each step in this flow in the following chapters.

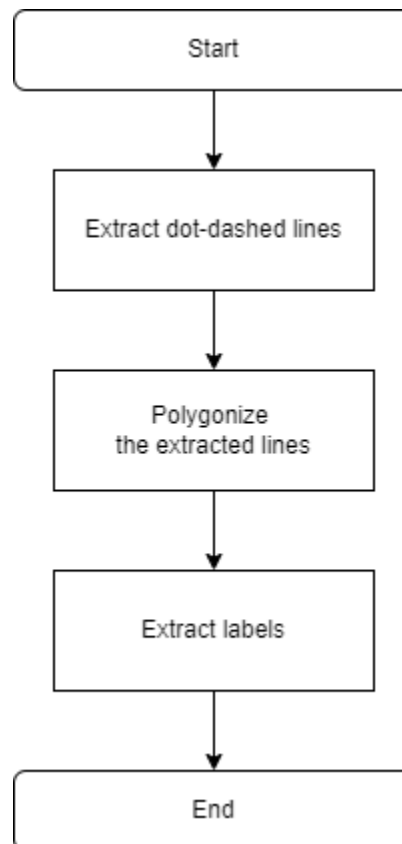


Figure 11. Main flow of the algorithm

### Dot-dashed line extraction

The extraction of dot-dashed lines is the most important task of the program. The accuracy of it heavily impacts on properly detecting each forest area. If even a small part of the line is not detected, an entire area can be not properly detected as a result since the area is the polygon fully enclosed by the detected line. Figure 12 displays that the flow of the dot-dashed line extraction algorithm. Throughout the algorithm, *shapely*, which is a Python package for manipulation and analysis of planar geometric objects, is heavily used to manipulate geometric vector objects such as lines and polygons.

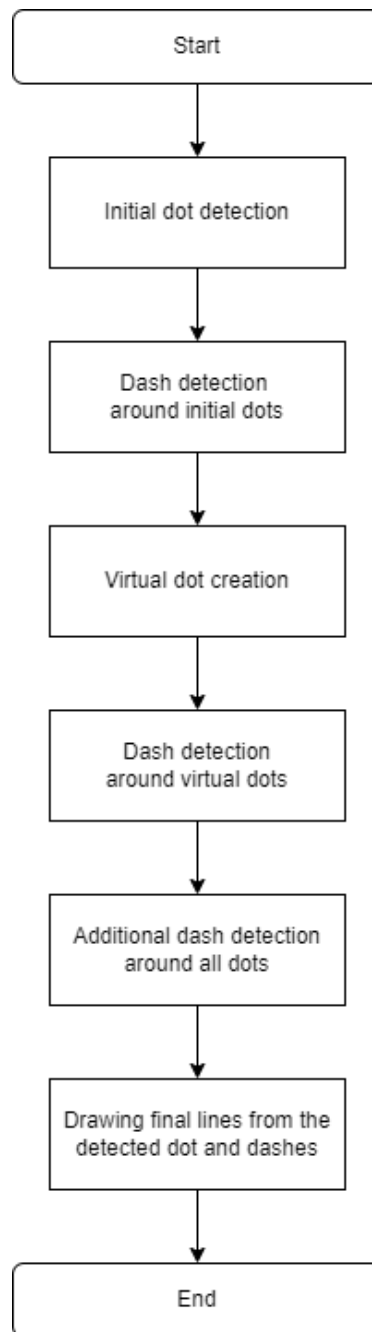


Figure 12. Flow of the line extraction algorithm



### *Initial dot detection*

The algorithm first detects dots of dot-dashed lines on the map. Mainly the algorithm uses an `OpenCV` class called `SimpleBlobDetector` and its functions to detect a certain size of blobs that are assumed as dots of dot-dashed lines.

### *Dash detection around initial dots*

Now that the algorithm knows the location of the dots initially detected, the next step is searching for dashes on dot-dashed lines. The algorithm first converts connected regions of black pixels into vector polygons using a `RasterIO` function called `shapes`. After that, the algorithm iterates the locations of the detected dots and searches for polygons around them. However, the polygons around dots can be something other than dashes. To prevent it, the algorithm uses several rules to filter out non-dash polygons.

### *Virtual dot creation*

Since forest cover maps are hand-drawn, there can be some degree of inconsistency in drawn symbols. For this reason, the dots on dot-dashed lines are sometimes not detected or even merged into near dashes. To handle this kind of cases, the algorithm creates virtual dots when the detected dashes do not have the dots detected near their endpoints.

### *Dash detection around virtual dots*

As new dots are created in the previous step, the algorithm tries to find dash polygons around the newly created dots.

### *Additional dash detection around all dots*

Even if the algorithm searched dash polygons, there are some special cases where dash polygons cannot be detected in the previous steps. Therefore, the algorithm searches dash polygons again but with different rules.

### *Drawing final lines from the detected dot and dashes*

Finally, vector lines are created using the detected dashes and dots. The final lines are obtained by using `networkx` library. With `networkx` library, the centerline of a dash polygon and the connecting lines between the dash polygon and the dots near it are converted into a graph. After that, `networkx`'s `shortest_path` function is used to get the final lines of a dash polygon and its nearby dots.

The reason the final lines of a dash and its nearby dots are created this way is to prevent overlapped symbols or other types of lines drawn. For example, a long solid line can cross a dash. In this case, the solid line can be regarded as an extremely long dash. However, our algorithm only considers the paths between the dots nearby a dash polygon as valid lines, this long solid line is not included in the final lines.

### *Polygonizing the extracted lines*

Once dot-dashed lines are properly extracted and there is no gap in the lines, the areas enclosed by those lines can be easily extracted by using `Shapely`'s `polygonize_full` function.

### Label extraction

In terms of extracting labels on the map, our program relies on *Keras-OCR* Deep Learning framework which is specialized for Optical Character Recognition (OCR). *Keras-OCR* provides a default model pretrained by their datasets. However, this pretrained model does not perform well on historical forest cover maps since it is not fine-tuned to the characters on the maps.

Therefore, to improve the accuracy of the model, we gathered our own dataset, which consists of more than 1200 samples of labels from forest cover maps, and trained the model with it. Figure 13 shows an example of training sample. If a developer wants to gather more training data and improve the model we trained, they can refer to the part of *fine-tuning the recognizer*<sup>3</sup> in the documentation of *Keras-OCR*.

The FMS tool uses our trained *Keras-OCR* model to recognize labels from the forest cover map given as input. Once the recognition is done, *Keras-OCR* provides the recognized characters and their pixel coordinates on the image given as input. The program transforms the pixel coordinates of the recognized labels into the geospatial coordinates based on the geospatial information in the image file (i.e., GeoTiff) given as input. By using these geospatial locations of labels, the labels are associated with the polygons extracted in the previous step if the centroids of the labels are within the polygons.



Figure 13. An example of training sample

---

<sup>3</sup> Keras-OCR, fine-tuning the recognizer

[https://keras-ocr.readthedocs.io/en/latest/examples/fine\\_tuning\\_recognizer.html](https://keras-ocr.readthedocs.io/en/latest/examples/fine_tuning_recognizer.html)

## Parameters of the algorithm

These are the parameters for fine-tuning the algorithm of the FMS tool in `settings.py` file. The default values of these parameters are mostly obtained empirically.

- Dot detection related parameters
  - **MIN\_BLOB\_AREA** – Minimum area of connected regions of black pixels that is regarded as dot
  - **MAX\_BLOB\_AREA** – Maximum area of connected regions of black pixels that is regarded as dot
  - **DASH\_DOT\_DIST** – This parameter determines the distance from the endpoint of dashes and virtual dots. It is used when the algorithm determines the location of virtual dots to be created.
- Dot-dashed line extraction related parameters
  - **MAX\_DOT\_LEN** – Maximum dot length (i.e., perimeter of a vector polygon that is regarded as a dot). This value is used to filter out polygons of dots when searching for dash polygons.
  - **DASH\_SEARCH\_BOX\_W** – The total width of the boxes for searching dash polygons
  - **DASH\_SEARCH\_BOX\_H** – The total height of the boxes for searching dash polygons
  - **SML\_DASH\_SEARCH\_BOX\_W** – The width of the *small* box for searching dash polygons. This *small* box is used for the additional search of dash polygons in the line extraction algorithm.
  - **SML\_DASH\_SEARCH\_BOX\_H** – The height of the *small* box for searching dash polygons. This *small* box is used for the additional search of dash polygons in the line extraction algorithm.
  - **ENDPOINT\_FILTER\_R** – The radius of the circle used to filter out redundant endpoints of centerline of dash polygons. It is common that the centerlines of dash polygon contain unnecessary branches when they are created, and due to these branches, there can be redundant endpoints of dashes. This value determines the size of a circle and if there are multiple endpoints in this circle, they are regarded as redundant except one.
  - **VDOT\_FILTER\_R** – The radius of the circle used to filter out redundant virtual dots. Normally, if a dot is not properly detected, the dashes around this dot try to create a virtual dot. In this case, multiple dashes end up creating multiple virtual dots for a single dot that is not detected. This value determines the size of a circle and if there are multiple virtual dots in this circle, they are regarded as redundant except one.
  - **SEARCH\_STEP\_DEGREE** – The rotation step of dash search boxes in degree. After each iteration of searches, the boxes for searching dash polygons are rotated around a dot by this value.
  - **MAX\_SWAMP\_SYMBOL\_LEN** – Maximum swamp symbol length (i.e., perimeter of a vector polygon that is regarded as a swamp symbol). This value is used to filter out polygons of swamp symbols when searching for dash polygons.
  - **MIN\_SWAMP\_ENDPOINTS** – Minimum endpoints that a swamp symbol has
  - **MAX\_DASH\_LINE\_LEN** – Maximum length of the final line representing a dash and its connecting lines to nearby dots. This value is used to filter out solid lines. If this value is

too small, more solid lines are regarded as dot-dashed lines. If this value is too large, some dot-dashed lines are regarded as solid lines and filtered out.

- **MAX\_D2D\_DISTANCE** – Maximum dot-to-dot distance. If the distance between two dots with a dash in between is too far, it is likely a solid line crosses this dash and connects those two dots. This value is used to filter out this kind of cases.
- **INTERPOLATION\_DIST** – This value is used as an argument of a function called *Centerline* of *centerline* python package. The higher value, the less branches on the created centerline.
- **CENTERLINE\_BUFFER** – This value is to simplify a polygon before its centerline is created from the polygon. The reason to simplify a polygon is to have less branches on the created centerline. The higher the value, the bigger the polygon.
- **SIMPLIFY\_TOLERANCE** – This value is used to simplify the centerline created by a polygon. The higher value, the simpler centerline.
- **IMAGE\_BBOX\_BUFFER** – Image bounding box inside buffer value. The larger the value, the bigger buffer inside the image bounding box. This image bounding box is used to find intersection points between the edge of the image and the dot-dashed lines.

### Major issues needed to be resolved

As we mentioned earlier, the accuracy of line extraction algorithm is important. Even a small gap between the extracted dot-dashed line can make the program fail to extract a large area. There are some issues that the algorithm cannot handle properly.

### Finding the optimal parameter values

Overall, the accuracy of line extraction algorithm can be improved by adjusting the parameter values of the algorithm. However, the problem is, if we adjust the parameter values to relax the conditions for detecting dashes, many unwanted lines would be drawn, and therefore the final areas would be divided into many pieces.

On the other hand, if we adjust the parameter values to make the conditions stricter, some valid dot-dashed lines would be not drawn, and therefore the polygon is not extracted due to the gaps between lines. We are not sure there is a set of optimal parameter values to cover every cases. More testing may be required to resolve this issue.

### Dot-dashed line merging into solid line

As we mentioned in the previous algorithm section, our line extraction algorithm can filter out solid lines when they cross dashes. However, there is a case that a solid line should not be ignored. If a dot-dashed line merges into a solid line as shown in the center of Figure 14, the dot-dashed line cannot make a polygon because the solid line is ignored. A new logic that covers this case should be devised and included in the algorithm to resolve this issue.

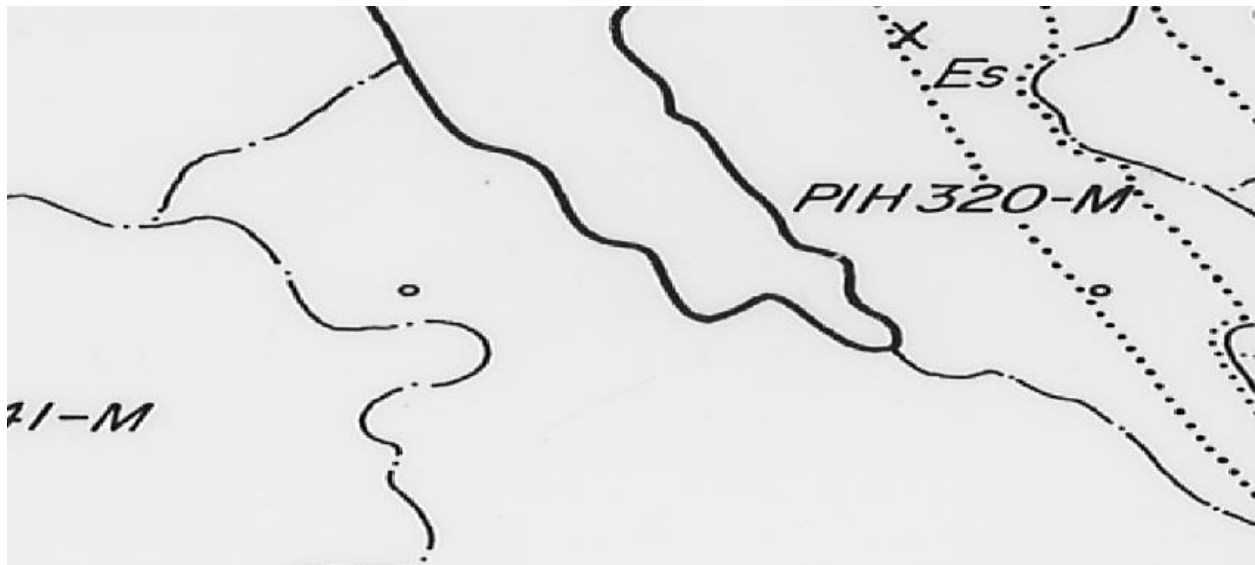


Figure 14. Dot-dashed line merging into solid line

### Symbols overlapping with dot-dashed lines

There are cases that symbols or characters overlap with dot-dashed lines. In most cases, they are ignored as we filter out solid lines. However, there are some tricky cases as shown in Figure 15.

In Figure 15, the word 'Creek' overlaps with a dot-dashed line, but it leaves small gaps around the overlapping regions. In this case, line extraction algorithm cannot draw a proper line on the dot-dashed line because the line is intermittent and some dots are absent due to the word 'Creek'.

There is another similar case in Figure 15. At the bottom of Figure 15, the word 'B.C. 4350' overlaps with the dot-dashed line. There can be more cases like these where symbols or characters overlap with dot-dashed lines. A new logic that covers this kind of cases should be devised and included in the algorithm to resolve this issue.



Figure 15. Overlapping symbols