Dalhousie University

ENGM 4620 - Python for Engineers

TAF File Updating Program Report

Project 1

Submitted: Wednesday, February 28, 2024

Submitted to: Issam Hammad

Executive Summary:

Laser cutters are commonly used machines, especially in the sheet metal manufacturing industry. One of the most popular and best brands for lasers is Trumpf. Granby Furnaces uses a Trumpf TruLaser 3030 to manufacture all unpainted sheet metal parts for their furnaces. One of the most time consuming parts of the programming process for a CNC laser is the nesting process. Nesting is when the flattened part files for different parts are placed on a virtual sheet of metal using a computer. This only ever has to be done once and then the sheet can be machined and run on the laser cutter as many times as needed.
In an ideal world once a part was made it would never have to be changed. However, this is not the case in reality. Especially during the design process, different parts may have many different revisions made to them. Every time this is done the nesting file has to have the all the parts inside of it deleted and readded to reflect the new revision. This can be very time consuming, especially when it has to be done multiple times for many different nests. The program that this report looks at aims to simplify this process by using Python to automatically run through nesting files and update the part files they point to. This would remove the need for a user to manually update every nesting every time a revision is made.

Table of Contents:

1. Introduction

This program can search through a directory and update specific (or all) .TAF files. TAF files, used by Trumpf CNC laser cutters, contain lists of parts (stored in .GEO files) and their locations on a sheet of metal, known as a nest. There is currently no easy way for the software provided by Trumpf to update the .TAF files if the name of a .GEO file changes. This causes issues and wastes time when part revisions are made. Granby Furnaces (the company this program was designed for) handles revisions for part (.GEO) files by changing the last character (or two) in the file name. For example, part number CAB-P1-0001-00 revision A is stored as CAB-P1-0001-00_A.GEO. If a minor (or major) modification is made to the design of that part, the revision increments to B, and the part file is now called CAB-P1-0001-00_B.GEO. With most minor modifications, the position of the part on the nested sheet doesn't need to change. Therefore, going into every TAF file, deleting the old revision off the sheet, importing the new file, and then saving the TAF file is time-consuming and pointless. TAF files are just specifically formatted text documents, so updating them simply involves finding each instance of the specific GEO file inside of them and updating the path to point to the file with the new revision.

2. Design Process

The design process for the program first involved figuring out a way to interact with TAF files using Python. Since TAF files are just text files they can be opened as such with a Python script. A manual test to see if changing the part file path inside of a TAF file was then done. This was to ensure that simply updating the file path was going to change the part file that the TAF files referenced. After this a set of features for a program that automates this was laid out.

These features fell into two main categories. There were TAF file handling features and config file handling features. These were what the FileManger and ConfigManager classes were built from in the program.

Once a method for each feature had been implemented, the main part of the program was constructed. It was decided that the main program should run on a loop, this removed the need for users to manually open the program for each new part that they wanted to update. Each one of the methods in the class was incorporated into the main part of the program. Console input was used to get any user defined values. This is a simple interface that can be upgraded using a GUI library (such as Tkinter) later if required.

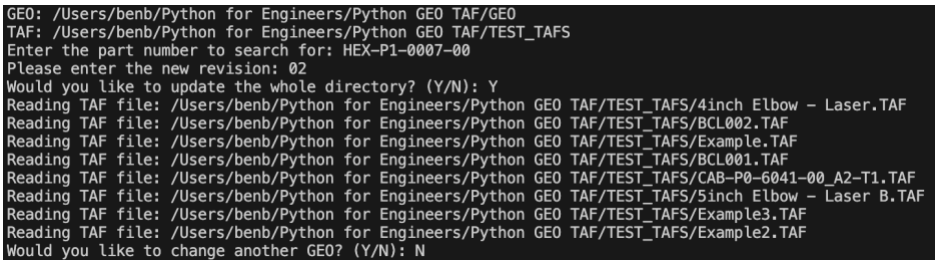

Figure 1: User interface example

2.1. Required Features

- Update all .TAF files in directory that contain a .GEO that matches the users request
- Update only specific .TAF files to new revisions.
- Get TAF and GEO directory from configuration file on startup.
- Save a log of all modified files.
- Ensure a GEO file with the requested revision exists. Ask the user to confirm if no match is found.
- Work between design and released for production parts. Design parts have a revision format of two numbers (i.e. CAB-P1-0001-00_02). Released for production parts have a revision

format containing letters (i.e. CAB-P1-0001-00_A). Also, should be able to move a GEO between design and released for production formats.

- Work for the part types of HEX, CAB, ELB. Part numbers will always follow the structure of XXX-X#-####-##_REVISION (X represents letter or number, # represents number)
- Be system safe (work on both Unix and Windows operating systems)
- **(MANDATORY)** Program should store files in temp files while iterating through lines. This is to ensure that if it gets killed it does not corrupt the files.
- *(Optional)* The ability for the user to replace the whole part number only in specific files, NOT THE WHOLE DIRECTORY!!!! (this might lead to the destruction of the entire nesting database).
- *(Optional)* The ability to store backups of modified TAF files for each batch of changes. These should correspond to an entry in the log file.

3. Program Structure

The program is structured using two main classes. It uses a FileManager class to interact with the GEO and TAF files. It also uses a ConfigManager class to interact and get the data from the configuration files.

3.1. ConfigManager Class

The ConfigManager class is structured with 4 methods: load_config, get_geo_dir, get_taf_dir, and get_backup_dir. Load_config loads the configuration file into the class. The other 3 methods use regular expressions to get values out of the configuration file for the GEO, TAF, and backup directories.

3.2. FileManager Class

The FileManager class contains one main method:read_and_update_taf_files
This method calls a variety of other methods within the class. These allow it to perform a variety of functions, such as backups, searching lists, and creating a change log. The main method also relies on regular expressions to search for patterns that match the part number format in the TAF files. When it finds a match, it uses regular expressions again to pick out the section of the part number that is a revision and replace it with the new one.

4. Testing Process

A variety of tests were performed on the program to ensure it operates properly.

4.1. Testing Scenarios

Missing Configuration File
Test Case: The program was run without a config.txt file present in the directory.
Outcome: Error message received: "The configuration file config.txt was not found."

Missing Lines in Configuration
Test Case: The BACKUP_DIR line was removed or corrupted in the config.txt file.
Outcome: Error message received: "BACKUP_DIR configuration not found or is invalid in the configuration file."

Missing GEO Files
Test Case: A .GEO file that does not exist in the GEO directory was specified in the update request.
Outcome: Prompt received: "The GEO file does not exist, are you sure you want to update? (Y/N):"

Invalid GEO and/or TAF Directory
Test Case: The GEO directory path in config.txt was misconfigured to a non-existent location.
Outcome: Error message received: "The GEO directory /Users/benb/Python for Engineers/Python GEO TAF/GEO was not found."

Missing Backup Directory
Test Case: A backup directory was not created before running the program.
Outcome: If BACKUP_DIR was specified but did not exist, the program created the new directory and proceeded with the operations.

Wrong TAF File Specified
Test Case: A .TAF file for updating that did not exist or was not found in the specified TAF directory was specified.
Outcome: Error message received: "TAF file not found."

5. Conclusion

The program created with the features discussed functions as expected. All required features were implemented in a manner is very versatile and upgradable. The only feature that was not included was the ability to modify entire part numbers. It was decided that the risk a user of breaking nestings with this feature was too high. The program was tested in a variety of abnormal cases, and it passed each test. The program was also tested in its designed workflow and successfully updated TAF files without breaking the nestings. There are some ease of life features that could be implemented in the future. These are things like a GUI or possibly an undo function. Overall this program will greatly speed up the revision part of the design process for Granby.