International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 11 Issue: 12 | Dec 2024 www.irjet.net p-ISSN: 2395-0072

Autonomous Utility Pole Identification

David Pesin[1], Elizabeth Piersall[1]

[1]Oak Ridge National Laboratory, One Bethel Valley Rd., Oak Ridge, Tennessee, 37831 USA

Abstract – *The implementation of small unmanned aerial systems (sUAS) for the purpose of powerline inspection is an emerging concept among utility companies. Such operations can produce a significant amount of useful data for the purpose of training different machine learning models or keep track of the integrity of electrical infrastructure. As such, this system is developed for the purpose of properly cataloging and leveraging this collected data.*

Key Words: *Powerline inspection, unmanned small aerial systems, machine learning*

1.INTRODUCTION

Using an sUAS has become an increasingly popular method for the purpose of patrolling power systems and routine visual inspections of utility infrastructure.[1] Equipped with a camera, the pilot can inspect a circuit manually as the flight is occurring. Most platforms with such functionality also offer the ability to record footage of the flight. In this case, an operator can inspect the footage after the fact, but that footage has far more uses than just retrospective analysis by an individual. Such footage can be used to train any machine learning model for the purpose of object recognition and component analysis, something that has significant utility and interest.[2] The difficulty is converting the video into a useful and usable format for such a task.

Fig-1: A compromised utility pole where the wood has rotted due to damage caused by burrowing woodpeckers.

The intent of this project was to lay the foundation for enhancing powerline inspection routines by creating a system for the future automation of initial circuit patrols following a detected fault. The goal is to develop a functional prototypical software that automatically parses a video from an sUAS overflight with its respective flight log and outputs a set of images tagged with all relevant information including GPS and attitude information, a timestamp, and a pole identification number if there is a utility pole present in the frame. This not only makes it immediately easier to use this data on its own, but also significantly enhances the ability to train a machine learning model off it. The system can be separated into four major components: data acquisition, data extraction and encoding, data filtration, and pole tagging.

Data acquisition broadly encompasses any overflight of a pole route that collects video footage. For our purposes, we are using platforms that leverage the open-source autopilot ArduPilot which can be run on a variety of flight controllers and multiple different sUAS form-factors such as multirotor drones, fixed-wing aircraft, and even blimps. Regardless of form-factor this software allows for the simple autonomous extraction of the logfiles after flights. In this case, the ArduCopter derivative is used. These sUAS also allow for simple video recordings of our entire flight to be saved as an MP4. Because we are using the same drones and cameras for all acquisition, the formatting of the videos and data is the same.

After the data is collected and extracted from the sUAS, video and telemetry information are parsed separately then combined. The video itself is split into frames using the Python package PyAV, while the logfile is parsed with assistance from PyMavlink. The extraction system can take any uploaded .MOV or .MP4 video file and separate it into either every frame or just the keyframes whilst simultaneously storing the uploaded logfile in memory after converting it into a usable format and deleting irrelevant information. This information is then processed together, matching each frame to a corresponding line in the logfile with the nearest timestamp. That logline is then parsed and linked to the frame itself. This is done by either using the Python package PyExiv2 to tag the EXIF metadata of each frame with relevant information, or by simply storing each frame index and relevant log line in a CSV file for future database storage depending on the utility's needs. The data is then filtered for the purpose of removing unhelpful data via an altitude threshold and a line detection algorithm.

Fig-2: A Holybro x500 v2 quadcopter frame equipped with a GPS, flight controller, telemetry radio, and a payload simulating the weight and shape of the camera system.

This encoded information can then be used for the purpose of creating a training set for any number of machine learning models. One that has shown significant promise is the You Only Look Once (YOLO) v8 model for its integration with the cloud-based tagging utility Roboflow and its efficient object-detection and component analysis capabilities [3]. The main drawback for such methods is that they require a significant amount of diverse and useful training data to produce consistently accurate results. Most openly available sources of data use public sources to find examples, such as using Google Maps. However, with this method one will be enabled to have datasets tailored for the specific infrastructure types used by the relevant utility so long as its use is consistent.

2. DATA ACQUISITION

Data can be collected for any purpose and on any mission from a camera equipped drone. Data acquisition can be implemented via a different sUAS as well. Consumer drone companies like DJI and Parrot can record their flight paths and some can even enact preplanned missions. However, for the purpose of this system, ArduPilot-based systems are preferred to keep flight log formats consistent. The sUAS can be piloted either manually or using a ground control system (GCS) such as Mission Planner. These systems enable compatible sUAS to follow a series of predetermined GPS waypoints and commands sent through a serial protocol known as MAVlink, allowing for repeatable missions to consistently collect data over the same region under different weather and lighting conditions. All the data collected to verify this process was done at various utility partner managed locations using a Holybro x500 v2 quadcopter mounted with a small, fixed camera connected to a Raspberry Pi recording at approximately 30 frames per second (fps) at 720p similar to the setup shown in Figure 2.

In this specific implantation the log can be extracted by removing the internal SD card of the flight controller upon landing the sUAS. These logs are automatically created upon arming the sUAS and are written to until the sUAS is turned off. There are also means to stream the log directly over a MAVlink telemetry port to expedite the process further. The video file is generally locally stored in the Raspberry Pi and can be extracted either remotely or physically. It is important to do both so that the logs can be organized by flight date and time in a directory. Afterwards, the directory itself can be sent to the extractor and encoder for processing.

3. DATA EXTRACTION AND ENCODING

The extractor and encoder take an ArduPilot logfile and a video file as inputs and outputs a directory of video frames with either custom metadata encoded inside of each or a CSV file containing relevant information. Figure 3 shows the simplified steps in this process.

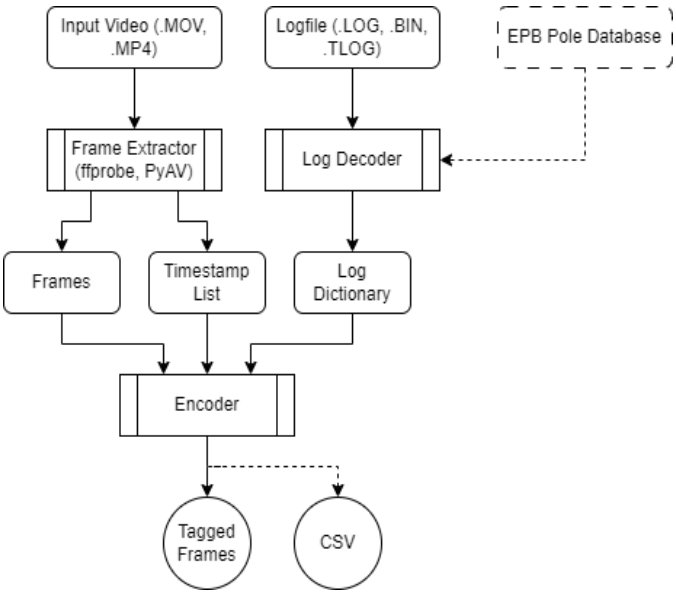

Fig-3: Flow chart showing inputs to each module of the encoder and their respective outputs

The first step is the frame extraction which has several alternative outputs available. First, a container for the video is created through PyAV where the software can open a video stream internally. Afterwards the start time of the video in epoch format is saved by extracting it via ffprobe while correcting for errors in the generated timestamps as needed. The stream is subsequently parsed through and only the keyframes (or i-frames) of the video are extracted, storing the timestamp of each frame by adding the frame time from the stream itself. Frame time is calculated via the video's framerate internally by PyAV. Keyframes represent a complete image within the video itself, while subsequent frames between them represent only the information that has changed from the previous keyframe. By only parsing the

amount of redundant data to be tagged is significantly reduced. As an example, one flight video used for testing contains 598.9 seconds of footage at approximately 29.97 fps. The total number of frames that can be parsed from this video is 17,949. From those frames, there are 1197 keyframes. While this significantly reduces the number of data produced, it also only extracts the most relevant information from the video to expedite the future tagging process.

Simultaneously, if provided, the logfile is parsed for relevant information. The logs themselves are created within the flight controller (FC) of the sUAS taking the recording. ArduCopter firmware provides a format to follow at the creation of these files, but the formatting may change depending on the FC software version or settings. To accommodate this, the first step is to parse the logs into a usable state is make sure that it is in plaintext. Files that use the .log format are usable as is, but if they are binaries or tlogs they can be put through the PyMavlink utility "mavlogdump.py" where they will be decoded. Formatting information is then extracted from the first few lines of those resulting logs. This step is both to allow us to decode the rest of the file, but also to determine if the file itself has enough information to tag the video. At the bare minimum logs should contain a timestamp and GPS information, otherwise they are not suitable for this step.

Next, the software parses through the entire logfile and extracts the relevant information including global location, relative location, attitude, and the epoch timestamp. In some cases depending on the FC settings, the timestamps are provided based on the connected ground control station's clock, but in most cases, they are simply the processor clock time in microseconds since the boot of the FC itself. In that case, the timestamp must be calculated via the information given to us by the GPS. This is done via the following equation.

$$Epoch = 315964800 + (604800 \cdot GWk) + (\frac{GMS}{1000}) + 18$$

The timestamp is determined by adding the GPS week (GWk) in seconds, the elapsed milliseconds in the current GPS week (GMS) in seconds, and the number of leap seconds to the GPS start date in epoch format (01/06/1980 00:00Z). Because the GPS time is not updated with each entry in the log, gaps between those updates are supplemented by adding the difference between the current log line's FC clock time and the previous.

Each GPS coordinate in the parsed log is then compared to a database of utility pole locations and ID numbers and will pair the ID of the closest pole to each image. If the pole is greater than 25 meters away from the GPS coordinate, it is considered too far away to be identified in the image. provided us an example of such database for the purpose of testing.

With the logs are parsed and the frames are extracted, the software begins the actual encoding process. This is done by first aligning the start time of the video with its associated flight log. Afterwards the encoder pairs each frame with the line in the log that has the timestamp closest to the time associated with the frame in the list. At times the video is longer than the flight log or vice versa, and as such the software aligns the start times by checking which begins first and skipping the other to match it. In the example video mentioned prior, this brings the total number of applicable frames from 1197 to 1002. The pairs are then combined using PyExiv2, thereby encoding the GPS and pole information into custom XMP tags in the individual frames. Thus completes the process of converting a video and a log file into metadata encoded frames for storage and further processing. Another approach available in the software skips the metadata encoding process entirely and simply saves the index of the frame next to the relevant information into a CSV for future parsing.

Not every flight will produce a flight log. When using a proprietary drone without many customization options it isn't as simple as pulling a logfile directly from the flight controller. The videos on their own can still be used to train machine learning data, however the automated location tagging won't be possible.

4. DATA FILTRATION

This method produces a large amount of data, but not all this data will be useful for training. On top of that, the tagging process to create that training set must be manually done, and it is a time-consuming endeavor. To minimize the amount of human effort necessary for such training whilst also keeping the integrity of the data, the software can preprocess the set of frames for that purpose.

For instance, a video may contain a significant amount of footage of the drone before taking off which generally consists of a blurry close-up of the ground beneath it. The method of filtering out these images is as simple as setting an altitude minimum, thereby pruning any image where the drone either hasn't taken off yet or reached the target altitude. This is only possible if a flight log is created, but there are other means of filtering less important frames. From the 1002 valid frames provided, 646 of them were taken at an acceptable altitude, in this case 30 feet above ground level. This threshold can be adjusted based on the average height of the utility poles.

Another method of preprocessing the data is prematurely running it through a preprocessing algorithm such as a line-detection algorithm for the purpose of aiding the human tagger with custom tags. Components of power delivery infrastructure are primarily linear such as

conductors, crossbeams, and the utility poles themselves. The Hough line transform has been effectively implemented in systems for the detection of such components.[4] A similar process has been implemented for the purpose of filtering the frames extracted by the previous step using the open-source image processing library OpenCV. The following figure shows each step as it is done for the purpose of pole detection. The idea is to extract lines that are longer and more distinct than what would occur in anything in the background of the image. Given that powerlines tend to be sets of parallel lines within a cluster, this should be enough to determine that there is likely either powerlines or a utility pole present.

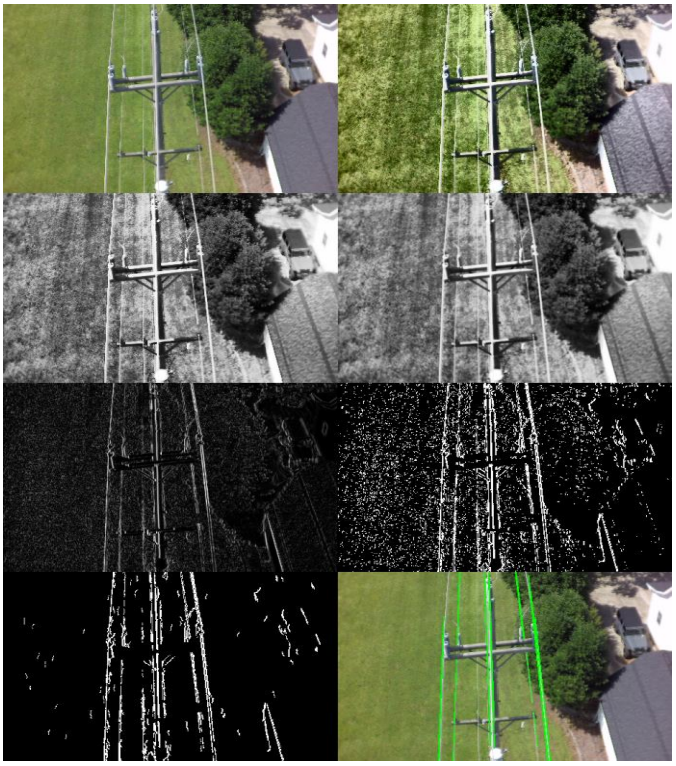

Fig-4: In order from left to right: Original image, contrast increase, grayscale, gaussian blur, Sobel y gradient, thresholding, component analysis, probabilistic Hough line transform.

The steps to this process are as follows:

(1) The contrast of the image is increased by splitting it into LAB color space and applying contrast limited adaptive histogram equalization (CLAHE) onto the L-channel, which represents the lightness in the image. This is done to improve the visibility of edges in images with poor lighting or noisy backgrounds.[5]

(2) The image is turned to grayscale, as the edge processing requires it as an input.

(3) A Gaussian blur is applied with a 7x7 kernel to the image to reduce excessive noise and display features more prominently.

(4) Sobel edge detection is performed on the resulting image, only focusing on the y-gradient. This means primarily vertical edges will be highlighted in the resulting image. It is possible to implement both edge detection methods to include horizontal lines in the case of footage where powerlines are consistently perpendicular to the poles; the above example focuses on overhead shots with the sUAS following the conductors.

(5) A threshold is then applied onto the image to make the edges more cohesive and convert the image into a binary for the transform.

(6) Component analysis is performed on the remaining artifacts that do not exceed a given minimum area and height to further highlight the cohesive lines in the image.

(7) Lastly, a probabilistic Hough line transform (PHLT) is performed on the binary image to detect consistent segments of straight lines in a group, thus highlighting the structure of a power pole. This method takes the standard Hough transform and applies restrictions such as a minimum segment length and maximum distance between segmented lines to further filter the output.[6] To prevent false negatives, a non-probabilistic Hough line transform can be applied in its stead.

This method allows us to detect the region where a power pole may be present and pre-tag any images that do not contain a single line as not containing poles. Images without any lines are generally either too noisy or blurry to be a positive example of a pole or occur when the drone is flying over a region with no manmade structures. Regardless, the data should be reviewed manually during the tagging process using this filtration step as a guideline, as this method requires the footage to not be blurry and to have enough contrast to produce an edge between the background and the components in question.

5. POLE TAGGING

After this process the videos have been transformed to a set of images tagged with pole locations and other relevant information pertaining to them. The next step is to take that information and tag it for the purpose of training a machine learning model on it. As an example, the data collected in this research will be used for training a YOLOv8 model to detect the presence of a utility pole and separate the pole itself from the rest of the structure. This work utilized Roboflow, a web-based tagging tool built by Ultralytics to simplify the process. This is a system that has been explored for various industries, and there are datasets publicly available on the platform for training and testing relevant models.

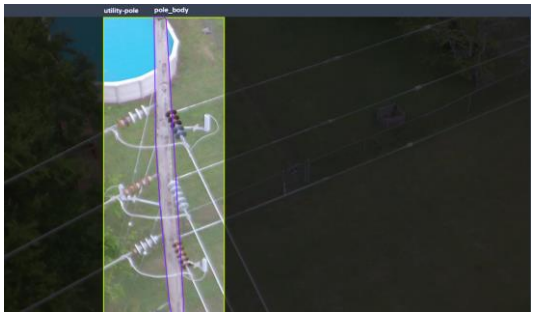

Fig-5: A tagged frame as it appears in the Ultralytics Roboflow web application.

The full dataset produced by this project is comprised of 3,787 frames extracted from footage of five locations. From the raw footage, approximately 35% of the frames contain an image of a utility pole, meaning most of the training data contains negative examples. This is intentional, as there are many objects that are easy to misidentify as power poles such as light posts and trees. 70% of the frames are used to train the model, 20% are used for validation, and 10% are used to test. As shown in Figure 5, we have two object classes we are seeking. The "utility-pole" class is denoted by a bounding box while the "pole-body" is denoted via a more complicated polygon for segmentation. This allows us to train a model through the YOLOv8 instance segmentation model. This process, along with the previously tagged locations and timestamps is the final step for the conversion of raw flight footage into a useful training set.

6. EXAMPLE MODEL

Roboflow allows for conversion of uploaded datasets into formats conducive for training a multitude of different models, including the aforementioned YOLOv8 component segmentation model. This dataset used to train a model in 100 epochs with no preprocessing done to the images beforehand to get an understanding of how effective the data currently is. Given the variety of locations and data quality, the results were acceptable despite leaving room for improvement.

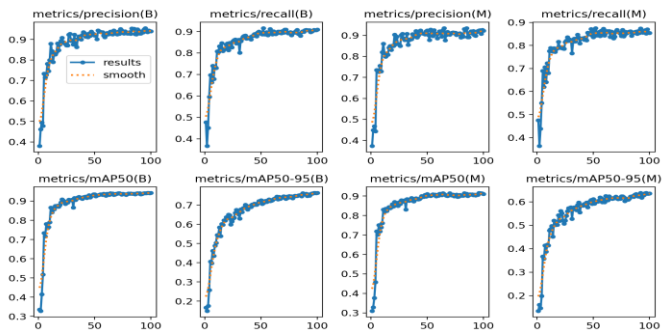

Fig-6: The set of graphs showing the improvement in different commonly used object-recognition metrics over the course of 100 training epochs

This model can accurately determine the general presence of a utility pole and its appended components along with segmenting the central pole itself. Figure 6 contains graphs showing us different metrics for the model trained with the given data. Graphs containing '(B)' refer to the bounding boxes surrounding the utility pole itself while graphs containing '(M)' refer to the mask around the structural pole. Two of the metrics contained in the figure are precision and recall. Precision (P) refers to the number of true positives (TP) out of all positives including false positives (FP) a model returns, while recall (R) refers to the number of true positives compared to the number of false negatives (FN) a model returns.

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

The precision and recall of the bounding boxes are 0.956 and 0.916 respectively, and the values for the masks are 0.935 and 0.876. This displays that even with the diverse and suboptimal data provided, a functional and accurate model has been trained.

Using those values, a metric known as the mean average precision (mAP) is calculated by first calculating the average precision (AP) of our model. This is area underneath the precision-recall graph[7]. The mAP is the average of all classes under consideration (which in this case is two) under different intersection over union (IoU) thresholds. This threshold determines what our model considers a false positive versus a false negative. The more the threshold approaches 1, the more accurate the predicted boxes or masks must be to the validation set.

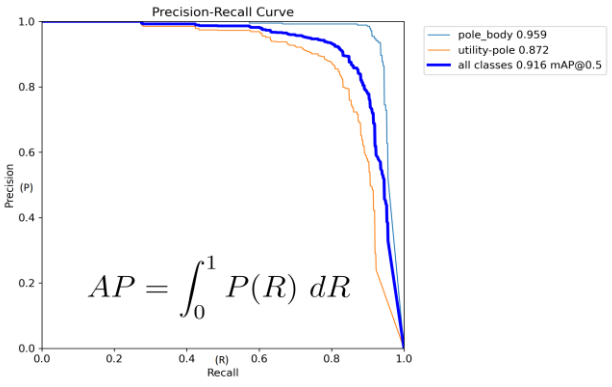

Fig-7: The precision recall graph for masks along with the formula for average precision.

The maP for the box and mask is relatively high at 0.94 and 0.91 respectively when the IoU threshold is 0.5 meaning the model can make accurate predictions when only expected to be half as precise as the validation data. However, the model begins to struggle when the IoU threshold is increased 0.95, the precision dropping to 0.76 and 0.64 respectively.

Fig-8: A set of accurate predictions made by the model.

7. CONCLUSION

With minimal preprocessing and improvements done to the dataset at large, the software was able to turn completely unused data into a format able to train a functional model for the purpose of pole detection and component segmentation. With every flight taken by utilities for the purpose of pole inspection, even more data is being collected that could be used to improve upon existing models through easily accessible methods. The training of such models will allow for the more accurate powerline inspection results than what could be done with simply a visual inspection, and by expanding on this tagging system with more detailed object classes it is possible to create more specified and useful models as well. For instance, training a model for the purpose of extracting insulators and differentiating insulator types can be useful for the detection of cracked or otherwise anomalous insulators. Spending more time on the preprocessing and filtration of the collected data may also further improve the efficacy of any models trained from newly collected and previously existing data.

REFERENCES

[1] Wang, Z., Gao, Q., Xu, J., Li, D. (2022). "A Review of UAV Power Line Inspection." In: Yan, L., Duan, H., Yu, X. (eds) *Advances in Guidance, Navigation and Control . Lecture Notes in Electrical Engineering*, vol 644. Springer. https://doi.org/10.1007/978-981-15-8155-7_263

[2] A. Varghese, J. Gubbi, H. Sharma and P. Balamuralidhar, "Power infrastructure monitoring and damage detection using drone captured images," *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1681-1687, https://doi.org/10.1109/IJCNN.2017.7966053

[3] Redmon, J. "You only look once: Unified, real-time object detection." In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[4] T. W. Yang et al., "Overhead power line detection from UAV video images," *2012 19th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Auckland, New Zealand, 2012, pp. 74-79.

[5] R.A. Manju, G. Koshy, Philomina Simon, "Improved Method for Enhancing Dark Images based on CLAHE and Morphological Reconstruction", *Procedia Computer Science*, Volume 165, 2019, pp. 391-398, https://doi.org/10.1016/j.procs.2020.01.033

[6] N. Kiryati, Y. Eldar, A.M. Bruckstein, "A probabilistic Hough transform", *Pattern Recognition*, Volume 24, Issue 4, 1991, pp. 303-316, https://doi.org/10.1016/0031-3203(91)90073-E

[7] Zhu, Mu. "Recall, precision and average precision." *Department of Statistics and Actuarial Science, University of Waterloo*, Waterloo 2, no. 30, 2004, p. 6. https://datascience-intro.github.io/1MS041-2022/Files/AveragePrecision.pdf