TACTILE: Tactical Analysis and Communication through Text Integration with Language Engines

By

Jack Stevens

_____

A Thesis Submitted to The W.A. Franke Honors College

In Partial Fulfillment of the bachelor's degree
With Honors in

Statistics / Data Science

THE UNIVERSITY OF ARIZONA

M A Y 2 0 2 4

Approved by:

Dr. Marek Rychlik
Professor of Mathematics

Abstract:

TACTILE (Tactical Analysis and Communication through Text Integration with Language Engines) implements advancements in OCR techniques and integrates a Large Language Model to extract and communicate text from images decisively. This study intends to mitigate bias and distrust within dual-AI technologies and develop AI-integrated software for text analysis. This software will have commercial and military applications to extract, interpret, and extrapolate image data. This project is guided toward developing drone software for military intelligence to capture and communicate text found on images into compressed data packages discretely with a defined level of autotomy. The commercial aspect of this software would entail the application of automated data collection from sensor readings and other forms of texts found in daily life. This project aims to demonstrate the connection between the LLM, deep learning, the natural world, and the limitations of pre-trained models. Developing algorithms that can operate within limited computation power and maintain accuracy and efficiency.

Introduction:

This project was performed in collaboration with Marek Rychlik and David Ryan and funded by the FY24 NSS TRIF: Managing AI as Dual-Use Technology grant to demonstrate both military and commercial use for AI. I chose this project because it combines the subjects I have studied over four years at the University of Arizona. This project requires system planning, deep learning, simulation, and experimentation. Balancing the computation power of different devices to detect text in real-time from a drone was where most of my time was invested during this project.

Problem Statement:

The techniques and models implemented to investigate this objective were optical character recognition (OCR), text detection, and large language models (LLMs). Optical Character Recognition, or OCR, is a model that searches images for text. This study used Tesseract (*Tesseract)* as the OCR engine due to its accessibility in both Python and MATLAB. Text detection is the process of selecting an area that contains text. For this project, we explored the CRAFT (Baek, 2019) and EAST (Zhou, 2017) algorithms as two different text detection models. To incorporate the AI aspect into the project, the ChatGPT model was chosen as the Large Language Model to comprehend the text extracted from the images, assign context, and determine relevance to a given scenario. Since the OCR performs best on text that resembles printed text, image processing is required to achieve the best results. Both Python and MATLAB contain image processing packages.

There are two models in which a drone can be operated: the ground station model and the autonomous model. The ground station model would have a computer in the loop relaying commands to the drone via Wi-Fi or TCP protocol. This would be beneficial for testing and efficiency since most of the computation power would come from the ground station instead of onboard the drone, and the computer can monitor the information being relayed from the drone. However, the issue would be that there would be a limit to the drone's ability to travel away from the ground station because it would need to be connected to the same Wi-Fi as the ground station computer. The autonomous model would be the preferred model since there would not be a computer in the loop because most of the computational power would be going to the onboard computer and directing its acts. However, the limitations of this would be that since the drone has limited computation power, the inference made by both the OCR models and text detection models would be limited and slow if not implemented properly. Furthermore, the autonomous model relies on the drone to send the information to the computer to see the image interpretation results. As a result, the ground station model would be the preferred method for testing and implementation.

MATLAB Simulink model

MATLAB contains UAV and image processing boxes that can produce a simulation that would demonstrate the project's overall goal. In the simulation, the drone would fly semi-circularly while looking at the red text containing the word "Banner Health."

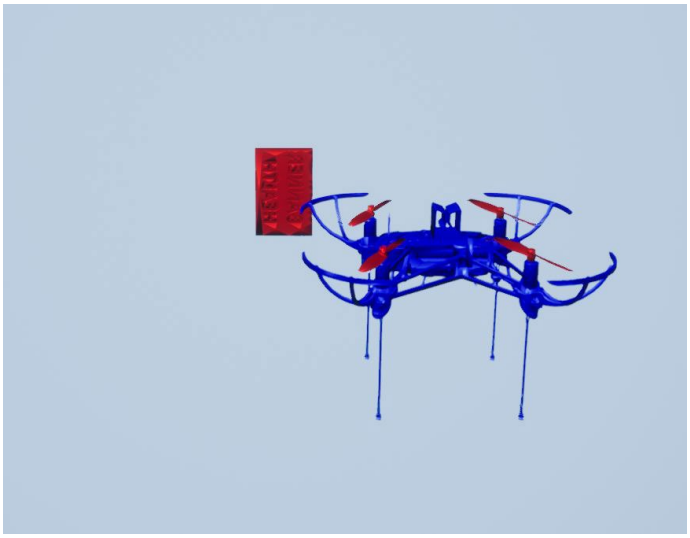

Figure 1: Adaptation of MATLAB Simulation UAV example developed by David Ryan.

This model demonstrated the need to continue with the ground station model for testing purposes before continuing with the autonomous model to define specific actions and information filtering requirements to pursue a more autonomous model. For example, in the model above, the OCR algorithm was called every time the drone moves in a circular motion. However, it may be more efficient to have the OCR called only when it is known that text is visible to the camera.

A limitation of this model was that no background noise was present to demonstrate how the drone would handle natural images. Furthermore, the text was presented in a way that the drone would always be able to read the text. In the real-world scenario, the drone would travel on a predetermined path and could detect text, fly towards it, and appropriately align itself with the text. To overcome this need for consistently calling the OCR algorithm, a text detection algorithm to find and encapsulate a box around the text, a 'bounding box' to help with the determination of the location of the text, is implemented.

Furthermore, implementing a method to send and receive information from ChatGPT was not supported through the current networking toolbox in MATLAB while using Simulink as the engine to run the model, and there needed to be filtering when a ChatGPT was sent information. To overcome some of these limitations, it was decided to establish a list of seen words within the script to ensure that the words sent to ChatGPT were unique each time. Furthermore, there needed to be a filter on the confidence associated with the word predicted and to neglect empty strings to help minimize background noise and false readings. Regarding implementation

requests for ChatGPT, these would require developing a MATLAB function outside of the simulation that would later turn into a MATLAB executable.

Text Detection

The implementation of the text detection algorithm was required to help the drone determine where the text was located within its camera view. MATLAB contains a different text detection algorithm within its image processing toolbox. The algorithm used was CRAFT text detection due to its flexibility in finding text. Since our target hardware would be an additional ARM chip like a Raspberry Pi, implementing the text detection, OCR, and ChatGPT API (*OpenAI, 2023)* Application Programming Interface) requests on the Raspberry Pi separately would be an effective way to test the capabilities on the drone's onboard computer.

However, after some experimentation, the text detection worked when the Raspberry Pi was connected to the computer using an ethernet cable. However, the limitation of this implementation was that the script could not be converted into MATLAB executable since the implementation of the CRAFT detection and the code used to make the ChatGPT API request did not support code generation.

As a result, it was decided to move the focus to Python since this is native to the Raspberry Pi OS. Using the OpenCV's (OpenCV)a machine learning library, a text detection model called EAST was accessible using Python. This model is known for its ability to detect text more effectively and with good accuracy when paired with GPU acceleration. This model is not as accurate as the Craft detection model when using the CPU computation power.

When testing this algorithm, it was found that it tends to underestimate the bounding box for the text. As a result, it was possible to achieve better results by expanding the bounding box slightly and cropping the image based on the bounding boxes. However, when reviewing the results of cropped images, in some test images, the resulting cropped images did not contain text, and the resulting OCR would have a short sequence of letters or punctuation that was not part of the image. As a result, the algorithm achieved better results by limiting the OCR only letters and then multiplying the length of the resulting text by the confidence of the boxing box to find a weighted confidence and filtering lower out lower values. After evaluating the algorithm, it determined that there had to be a standardized way of observing and capturing text to achieve better results. However, based on the motivation of using text detection in this project to determine if there is text in an image and plan a path toward it, the lack of accuracy was acceptable.

Figure 2: Image of Lebron James from the OpenCV GitHub repository for EAST Text Detection and Adaptation of EAST text detection. (OpenCV)

One limitation introduced through this text detection exploration was the speed at which these models could generate bounding boxes. This issue was emphasized with the EAST detection model. On average, a single image could be processed at a rate of 0.33 seconds per image. However, in the context of this problem statement, that would be slow since much of the drone path determination would be based on the bounding boxes produced through this model.

To address this issue, it was discovered that there was a way to implement the EAST detection model using a smaller model input through quantization and TensorFlow Lite, a submodule of the popular Python machine learning package (Paul, 2020). Quantization is the process of reducing the information stored within an image to a smaller data type. Some experimentation found that the INT8 datatype size had the fastest inference time. Through these changes, using a ground station, the reduced input size model minimized the inference time on the bounding boxes from 0.33 seconds to about 0.15 seconds. However, this quantization resulted in a reduction in the accuracy of the bounding boxes. This reduction in accuracy was within an acceptable margin of error since text detection in this situation is used to give the drone a general direction toward the text.

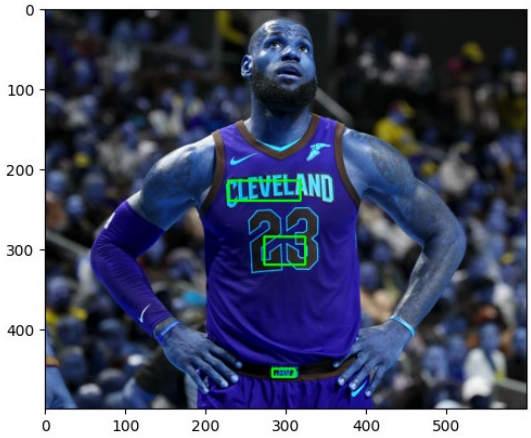

Figure 3: Resulting East Detection with Quantization of INT8 results. Code adapted from: https://github.com/sayakpaul/Adventures-in-TensorFlow-Lite/blob/master/ (Paul, 2020)

Gazebo Model

Since the goal is to have the drone react to the results from both the OCR and ChatGPT, a way the drone could be controlled through Python needed to be implemented. DroneKit (DroneKit) is a Python package that allows one to connect to a drone using Wi-Fi and control it using TCP by interfacing with the flight controller through the serial connections on the Raspberry Pi. Therefore, the Raspberry Pi must be wired to the flight controller, allowing the drone to have computation power onboard, eliminating the need for a ground station, and allowing the drone to act autonomously.

However, to develop a script to run the Raspberry Pi that would give the drone a level of autonomy, there needs to be a safe testing ground for experimentation. As a result, the Ubuntu environment containing ROS Neotric (noetic) and Gazebo 11 (Gazebo) was developed. By specifying the Ubuntu environment to have the exact hardware specifications as the Raspberry Pi, we could see how the Raspberry Pi would handle the created scripts and have a safe place to debug errors.

ROS is an open-source robotics toolkit that allows different information to be captured within a simulated world and then published to a console for analysis, like the camera feed from the drone model. ROS uses objects called nodes to publish data. To access the data, one can 'subscribe' to an active node and get the current information being captured in the simulation and then instantiate a new 'publisher' node that will add updated information to the ROS list nodes to see in the simulation.

In the case of capturing the image, the Quadcopter model developed by Drone Dojo (Berquist) and Ardupilot (Adrupilot) had already implemented a ROS node that published the raw camera feed from the drone. The results below were created by subscribing to the raw camera feed and

publishing a secondary feed that included the OCR bounding boxes that the PyTesseract (Lee, 2020) generates.

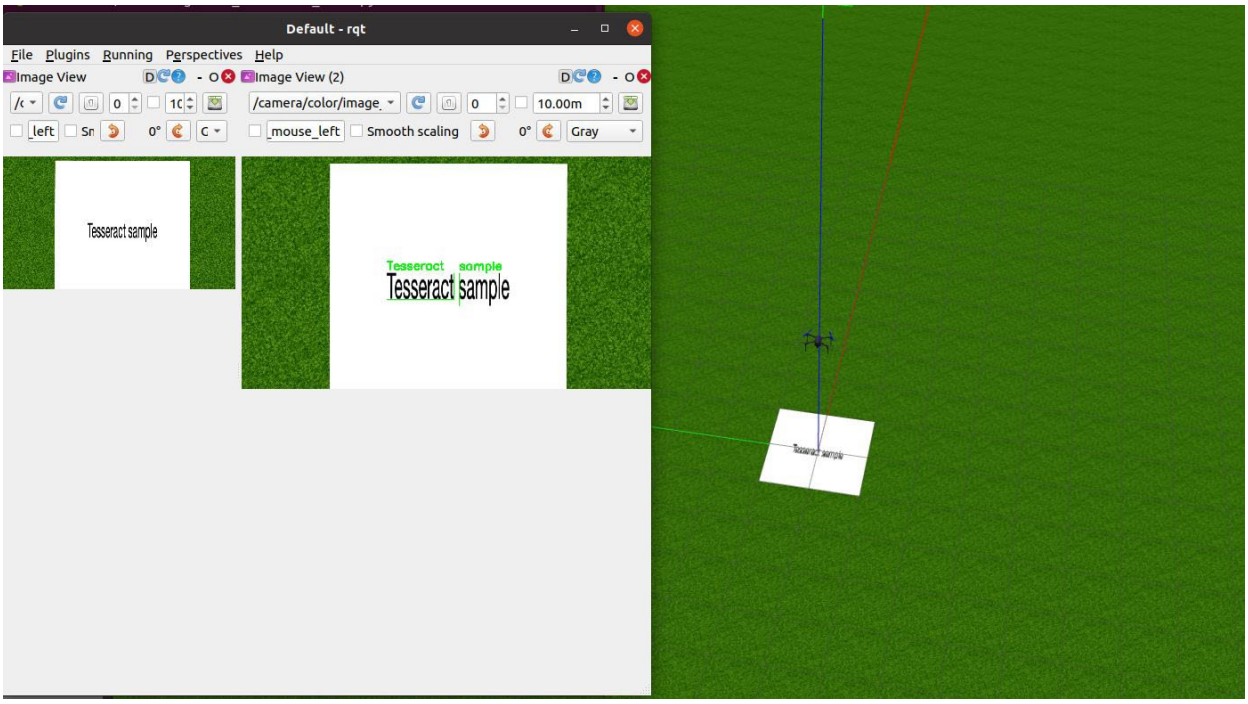

Figure 4: Gazebo using OCR Drone simulation. To the left demonstrates the raw image and processed image. To the right reflects the drone flying within the Gazebo world.

One assumption made in this model is that the drone is already aligned with the text. Furthermore, the text did not require preprocessing of the image. According to the problem statement, the drone must read the text as it travels along a designed path. However, in this case, the drone hovers over the text.

Webcam Test and Exploration

This experiment aims to simulate how the drone would distance itself from the surface that the text would be on and maintain the accuracy of the OCR. A webcam and a page containing the single word "STOP" in large font were used to simulate the drone camera and text of interest. The webcam would remain stationary, and the page would move along the view path of the webcam to demonstrate the drone move towards and away from the text of interest.

Since the OCR produces the best results when reading text that resembles text on a screen, it was determined that using printed text would be optimal for this experiment to avoid substantial preprocessing of the video feed and focus on determining the optimal distance. The webcam was used to simulate the drone view of the text and would allow the script to run on a computer instead of onboard a drone.

Running OCR on the video feed and finding the area within the bounding box of the text would allow for the capture size optimal text that the drone requires to read the text effectively, despite the size of the text in reality. As shown by the figure below, if the drone is too far away from the text, the area of the bounding box decreases beyond the lower limit threshold, and the command to the drone is to move forward. If the drone is within the distance, then the resulting command to the drone would stop since the area is within the upper and lower limit area thresholds. If the drone is too close to the text, the resulting command would be to move backward since the area has exceeded the upper limit threshold.

Figure 5. Demonstration of the results from the experiment with Webcam.py using the printed page with the word "STOP."

Furthermore, the red dot indicates the center of the bounding boxes. Since the video feed is set to be 320 x 320 pixels, the drone can direct itself to the center of the box by comparing the center of the viewpoint. Centering the drone on will help mitigate background noise and ensure that the edge of the view has cut off no letters.

One assumption made in this experiment is that the drone is facing in the same direction as the text. As a result, if the text were at an angle, the drone would not be directing itself toward the center of the text. Other assumptions were that the text would be close to horizontal. However, if the text were to be at a 45-degree angle, we would expect there not to be any text detected or a significant decrease in the accuracy of the OCR. Finally, it is assumed that the text size would be big enough to achieve the appropriate bounding box area.

As a result, to resolve some of these issues, incorporating text detection would help find text that is further away and more challenging to detect by the OCR and guide the drone until the drone is close enough to generate bounding boxes from the OCR. However, based on the nature of the Quantization model, the bounding boxes are not as precise as the bounding boxes generated by OCR. Therefore, the center point that the drone would be guided towards would move frequently. This error could be reduced by storing the average of the center point of the bounding boxes generated by the EAST detection model. Doing so would result in a slight change and move the point to the true center of the text. Furthermore, incorporating parallel threads would help increase the rate at which the results are produced from the East detection by increasing the number of frames being processed at a time.

Conclusion

Through this project, I helped develop some of the groundwork for allowing the drone to operate at autonomy at a close real-time analysis. I expanded my knowledge of ROS and Gazebos and how they relate to developing robotic applications and simulations. I also demonstrated my understanding of machine learning by adapting OCR and text detection models on images captured by a drone. Finally, I demonstrated systems planning by developing and constructing a drone and different simulations that would imitate the drone's behavior under certain conditions.

GitHub link to my code:
TactileOCR/East_Text_Detection (github.com)

References

Baek, Y., Lee, B., Han, D., Yun, S., & Lee, H. (2019). Character Region Awareness for Text Detection. *ArXiv:1904.01941 [Cs]*. https://arxiv.org/abs/1904.01941

Bergquist, C. (n.d.). *HomePage*. Drone Dojo. https://dojofordrones.com/

Gazebo : Tutorial : Ubuntu. (n.d.). Classic.gazebosim.org. Retrieved April 29, 2024, from https://classic.gazebosim.org/tutorials?tut=install_ubuntu&cat=install

Lee, M., & Hoffstaetter, S. (2022, August 17). *pytesseract: Python-tesseract is a python wrapper for Google's Tesseract-OCR*. PyPI. https://pypi.org/project/pytesseract/

noetic - ROS Wiki. (n.d.). Wiki.ros.org. https://wiki.ros.org/noetic

OpenAI Python Library. (2023, May 7). GitHub. https://github.com/openai/openai-python

OpenCV: Introduction. (n.d.). Docs.opencv.org. Retrieved April 29, 2024, from https://docs.opencv.org/4.0.1/d1/dfb/intro.html

Paul, S. (2020, April 29). *A Tale of Model Quantization in TF Lite*. W&B. https://wandb.ai/sayakpaul/tale-of-quantization/reports/A-Tale-of-Model-quantization-in-TF-Lite--Vmlldzo5MzQwMA

ROS/Introduction - ROS Wiki. (n.d.). Wiki.ros.org. https://wiki.ros.org/ROS/Introduction

Tesseract User Manual. (2023, May 4). GitHub. https://github.com/tesseract-ocr/tessdoc

Welcome to DroneKit-Python's documentation! (n.d.). Dronekit-Python.readthedocs.io. https://dronekit-python.readthedocs.io/en/latest/Drone

Welcome to the ArduPilot Development Site — Dev documentation. (n.d.). Ardupilot.org. Retrieved April 29, 2024, from https://ardupilot.org/dev/index.html

Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., & Liang, J. (2017). EAST: An Efficient and Accurate Scene Text Detector. *ArXiv:1704.03155 [Cs]*. https://arxiv.org/abs/1704.03155