Achieving Autonomous Drone Flight via a Hopfield Network Computer Vision System

Jovan Yoshioka
Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee
jyoshiok@vols.utk.edu

Dong Jun Woun
Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee
dwoun@vols.utk.edu

***Abstract*—Computer vision is vital to state-of-the-art, next-generation technologies, from autonomous vehicles to augmented reality. However, these technologies often use resource-intensive deep learning models to achieve this computer vision. Hopfield neural networks (HNN) are explored as a relatively cheaper, robust alternative by developing a custom pattern-based computer vision system applied to autonomous drone flight. After detecting a pattern via lightweight, accurate, and accessible OpenCV image processing, the HNN handles the pattern recognition, utilizing associative memory capabilities. The pattern detection algorithm was accurate and robust against differing backgrounds, bright and dark lighting conditions, and various pattern orientations. The pattern recognition algorithm could still recognize patterns with fifteen to twenty flipped bits, i.e., noise, out of one hundred bits. After successfully testing the computer vision system with an autonomous drone, it was ultimately concluded that HNNs are a feasible method to implement a robust, CPU-based computer vision system for real-time pattern-based applications. Improvements can be made regarding optimizing HNN parameters.**

Keywords—computer vision, Hopfield network, autonomy, drone

I. Introduction And Motivation

Computer vision is a fastly developing domain of computer science that concentrates on extracting information from images and videos. There have been many use cases of computer vision where state-of-the-art computer vision models have achieved autonomous vehicles, facial and iris recognition, augmented reality, etc. Despite these remarkable achievements, the computational power utilized for computer vision models has become a burden to computer vision development. The state-of-the-art computer vision models have been stagnated by computational power because they often use deep learning models that consume much time and computational capacity for training.

Researchers have explored scalable deep learning, where Rasley, Rajbhandari, Ruwase, and He have recently trained deep learning models with over 100 billion parameters [1]. Nvidia has achieved training the BERT model with 1024 NVIDIA V100 GPUs with sixty-six teraflops in sixty-seven minutes [2]. Despite the gains in computing power gains, they still require too much power and time. It is imperative to research the relevance of non-conventional pattern recognition methods like Hopfield neural networks (HNN) to train complex computer vision models.

HNNs are robust because they are recurrent, connected, and symmetric neural networks. Symmetric and connected neural network neurons impact the output neurons with weighted connections. Recurrent neural network neurons influence the output to affect the successive input to the following neural network. Hence, HNNs offer a robust optimization and pattern recognition method. Additionally, researchers and developers can train HNNs on regular day-to-day computers, but they can also train larger HNNs with greater computing capacity.

The overarching goal of the paper is to show the possibility of HNNs' robustness as a computer vision model. The report presents a computationally inexpensive computer vision system that uses an HNN model trained on a weak laptop processor with exceptionally robust pattern recognition. In addition, we provide a qualitative comparison of the computational power needed because a numerical comparison of the computational power required to train HNNs is beyond this paper's scope and will be future work for the HNN computer vision system.

II. Related Work

Building the computer vision system requires two primary components, pattern detection via image processing and pattern recognition via the HNN. Researchers have already explored computationally inexpensive methods for image processing.

Wang and Olson introduce AprilTag 2, a fiducial marker system used in a variety of real-time, mobile robotics applications to assist in vision-based localization, object detection, tracking, and more [7]. They describe implementing a bit error limit to enumerate all possible patterns and then storing them in a hash table for future constant lookup times. However, this requires tuning another parameter, the bit error limit, and constrains the robustness of the decoding. Our work takes heavy inspiration from AprilTags' image processing algorithm but implements HNNs in place of AprilTags' decoding for more automatic robustness.

Cyba, Szolc, and Kryjak contributed to control strategies and real-time vision-based navigation regarding autonomous drone flight [5]. Their research utilized gate-based drone detection and had two control strategies that used UAV coordinate or ArUco gates for coordination. Similarly, Irfan, Dalai, Kishore, Singh, and Akbar‘s research contributed to autonomous drone flight by performing autonomous drone pathing with a real-time visual feed [6]. Their methods inspired our method of detecting the input pattern. Ignatius Moses Setiadi, Fratama, Ayu Partiningsih, Rachmawanto, Sari, and Andono demonstrate the feasibility of using OpenCV in real-time applications through their vision-based traffic monitoring system [16].

Using the drone's onboard camera, Natarajan and his colleagues use hand gestures to control drones [13]. They discuss three primary considerations: gesture recognition, visual variability, and safety assurances. Despite some differences, e.g., they use Haar classifiers for their gesture recognition, their general workflow is similar to our workflow – recognizing the pattern from a frame of the drone's camera feed. Additionally, they provide several useful considerations when working with computer vision and drones, including varying visual conditions, e.g., dark/bright lighting and clear/cluttered background, and safety. The DJI Tello EDU drone used for this project has built-in automatic safety features, including automatic stabilization and failsafe protection [10].

Researchers have also researched a wide range of use cases for HNNs’ image recognition capabilities.

Soni and her colleagues used HNNs to implement facial recognition [11]. Their work demonstrates the feasibility of using HNNs for computer vision tasks, even with image distortion. Our project goes one step further and applies the HNN’s computer vision capabilities to a real-time drone application. The consistently positive success rates, even with image distortion, show robustness which is vital for real-time situations like when flying drones.

Bow demonstrates recognizing binary, grid-like images of letters and numbers using HNNs [12]. He represents the letter “C” with a binary array of either “-1” or “1,” representing whether or not a grid unit is filled. From there, he passes this binary array into the HNN to ultimately recognize the letter. This demonstrates how an image can be represented to be valid input to an HNN.

HNNs are inherently limited to the number of memories or patterns they can reliably remember. Keddous and Nakib propose a new architecture to increase the memory capacity by employing several HNNs in parallel, using genetic algorithms to optimize how the patterns are stored among the various networks [14]. Although we do not implement this in our work, we wish to make the drone perform a wider range of more sophisticated tasks. The drone will need to be able to process more patterns efficiently, enabling real-time computation, i.e., it cannot just increase the number of neurons, which will create overhead. Keddous and Nakib’s work can be applied to process more patterns in real-time applications.

Krotov and Hopfield’s research has utilized HNNs to train and test images for pattern recognition [3]. Their experiment tested with a vast memory capacity and test images, and they concluded that images with a high gradient have better performance results. Gan and Liu have also utilized HNNs and wavelet packets to remove noise from images and show how they could be used for pattern recognition [4].

III. Method/Approach/Experimental Setup

A. Workflow

The following workflow is visualized in Fig. 1. After connecting to the DJI Tello EDU drone via Wi-Fi, a command is sent to begin retrieving its video feed via a UDP connection. Upon receiving a video frame from the drone, the frame is input to the pattern detection algorithm that performs image processing and outputs a sequence of “-1” and “1,” representing black and white, respectively. This sequence is input to the pattern recognition algorithm, i.e., the HNN, that outputs a matching saved pattern and its corresponding command. Commands are pre-defined and utilize the Tello SDK 2.0 [15]. This command is then sent to the drone.

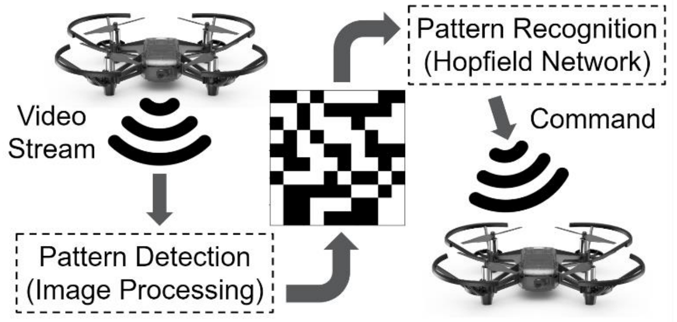
Pattern Recognition (Hopfield Network)
Video Stream
Command
Pattern Detection (Image Processing)

Fig. 1. Workflow Diagram

B. Pattern Detection

Pattern detection is accomplished through multi-step image processing. There exist novel methods to accomplish similar detections, such as AprilTags’ adaptive thresholding, continuous boundary segmentation, and quadrilateral fitting [7]. Although the concepts of such novel methodologies heavily inspire our image processing, we achieve pattern detection entirely through OpenCV, an open-source computer vision library [9]. Existing works have demonstrated OpenCV’s lightweight yet accurate performance in real-time applications, which are especially important factors when autonomously flying drones [8]. Additionally, unlike the aforementioned novel approaches, OpenCV’s common infrastructure enables greater accessibility to a wider range of developers for quicker and easier development.

Detecting a pattern from the drone’s video stream consisted of a low-cost, seven-part process, as seen in Fig. 2. The video frame is converted to grayscale, enabling the subsequent binary, adaptive threshold. Adaptive thresholding ultimately mitigates the effects of varying light conditions by using local thresholds instead of a global value for the entire frame [7]. A Canny edge detector is then used to extract the edges of the pattern, followed by contouring to convert these edges to a usable array of image coordinates. A pattern is finally detected as a contour with four corner points containing several contours inside of it, i.e., the internal grid units. Next, a perspective transformation is applied to this pattern for easier, more robust decoding, regardless of the orientation of the pattern, as seen in Fig. 3. This technique has superior performance when a region of interest (ROI) requires equal, uniform spacing [8]. Finally, the pattern is decoded in three steps. First, the size of each grid unit is calculated by dividing the ROI width and height by the number of grid columns and rows, respectively. Second, the centroid of each grid unit is estimated by iterating through each grid unit and calculating its image coordinate using the following formulas:

$$centroid\_x = column\ \# * unit\_width + unit\_width / 2 \quad (1)$$

$$centroid\_y = row\ \# * unit\_height + unit\_height / 2 \quad (2)$$

Third, the algorithm iterates through each centroid and determines if the corresponding grid unit represents a 0 or 1 depending on its color value, i.e., 0 for black and 1 for white. The bits are aggregated into an array that is then passed to the pattern recognition algorithm.

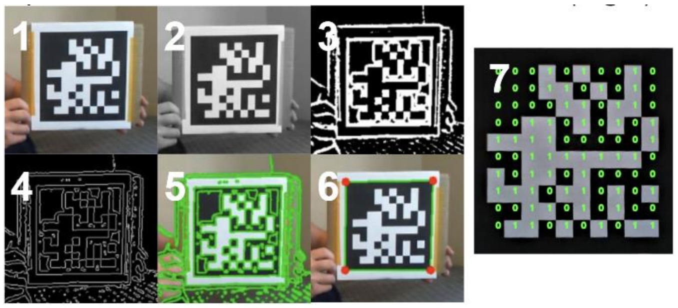

Fig. 2. Pattern detection process: (1) input frame, (2) grayscale, (3) adaptive threshold, (4) edge detection, (5) contours, (6) extracted pattern, (7) decoded pattern.

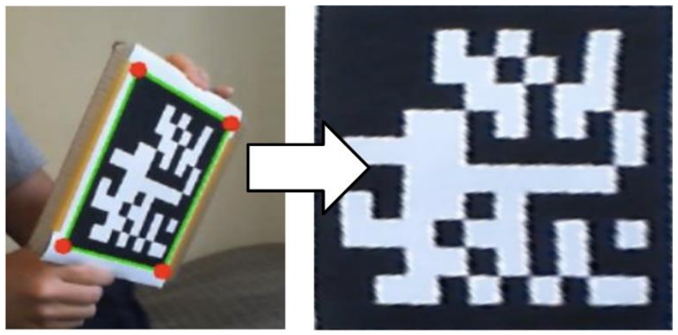

Fig. 3. Perspective transformation on an oddly-oriented detected pattern.

The pattern detection algorithm was tested with bright and dark lighting conditions, various pattern orientations, and three different backgrounds consisting of varying intensities of noise. The algorithm's performance against these factors was qualitatively evaluated. Patterns were also displayed at incremental distances from the drone's camera to quantitatively investigate the algorithm's range, assessing the detection and influence of background noise as the pattern moves farther away and becomes smaller. The number of false positives and false negative instances were recorded relative to the pattern's distance from the camera.

C. *Pattern Recognition*

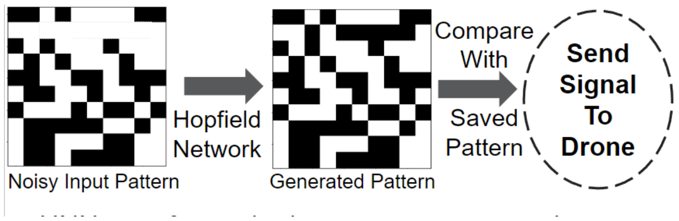

Fig. 4. Transforming a noisy pattern to a newly generated pattern.

The experimental detail our research entailed was the applicability of HNNs to apply a robust and lightweight HNN that researchers could utilize for autonomous drone flight. The experiment first aimed to determine whether pattern recognition was possible through HNNs. The project team timed the approximate time to recognize the patterns from the video feed to the drone command execution. The experiment secondly aimed to research HNNs' resilience to noise while staying computationally lightweight. The project team decided to recognize patterns through the HNN because the drone's video streams were reasonably accurate but could have been improved with analysis. A key value reviewed was the number of times an HNN needed its output to be input again to shift the input pattern towards an associated pattern for the number of noisy bits.

The HNN was implemented in Python, inspired by Dr. Catherine Schuman's HNN implementation. A set number of pre-defined patterns were imprinted onto the network. Initial weights of the HNN were randomly generated using NumPy. The pandas library was used to read and save randomly generated patterns and keep the error rates of the HNN. Finally, Matplotlib was used to generate graphs of the error rates and the networks used for training.

IV. Results

A. *Detection Results*

Designating a pattern as a contour with four corner points containing several contours inside of it, i.e., the internal grid units, allowed the image processing to successfully pick out the pattern and remove an abundance of background noise, as seen in Fig. 5.

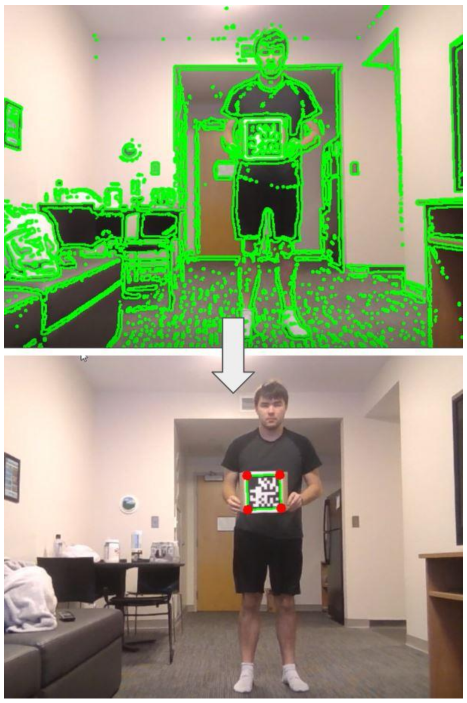

Fig. 5. Successful detection and removal of noise.

Unlike when extracting the pattern edges, adaptive thresholding was not useful in differentiating between the white and black pattern grid units in varying light conditions. As seen in Fig. 2.3, some white grid units become black, and vice versa, after applying the adaptive threshold. Instead, the minimum and maximum grid unit color values are found, and the average of these values is used as the white/black threshold. This method successfully decoded in bright and dark lighting conditions, as seen in Fig. 6.

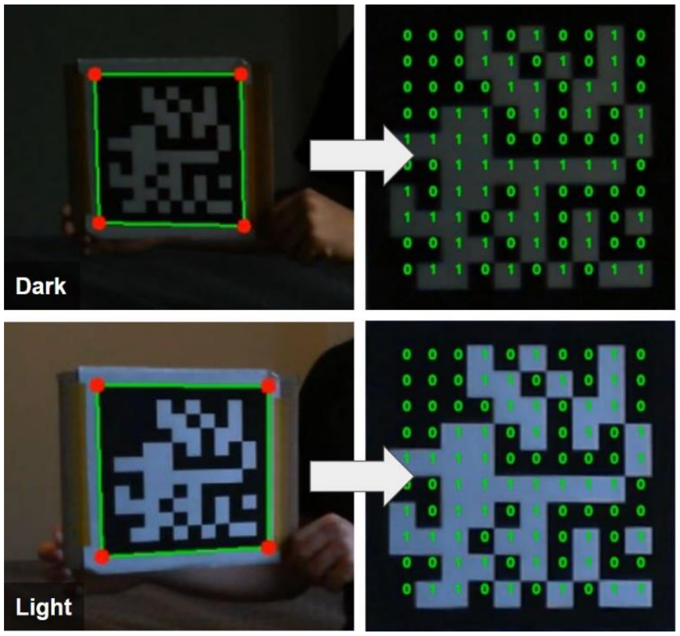

Fig. 6. Successful detection in varying lighting conditions.

The perspective transformation enabled the drone to robustly decode the patterns despite their orientation. We expected skewed orientations from holding the pattern when the drone was in flight simply due to human error. However, we did not anticipate how obstructed the drone's field of view would be when placed on the ground prior to taking off. The inconsideration required holding the pattern in the orientation seen in Fig. 7 to get the pattern inside the drone's field of view. Despite this unexpected obstacle, the detection algorithm was still able to detect the pattern.

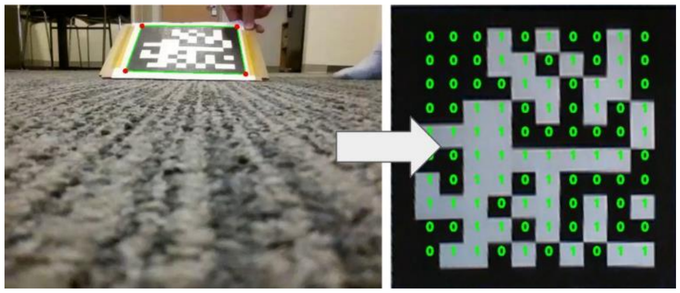

Fig. 7. Successful detection of a pattern with skewed orientation.

Although the algorithm performed well in the aforementioned ways, false positives and false negatives were apparent as the pattern was placed farther away from the drone's camera. After quantitatively evaluating the algorithm's range with three noisy backgrounds, as seen in Fig. 8, we found patterns from up to 8 feet away can be reliably detected, as apparent in Fig. 9 and Fig. 10. In Fig. 10, the absence of pattern detection as the pattern's distance from the camera increases is primarily due to the contour area threshold implemented to reduce the number of false positives. As the pattern moves away from the camera, it becomes smaller and eventually will not meet the area threshold. There is a tradeoff between the number of false positives, i.e., reliability, and the range.

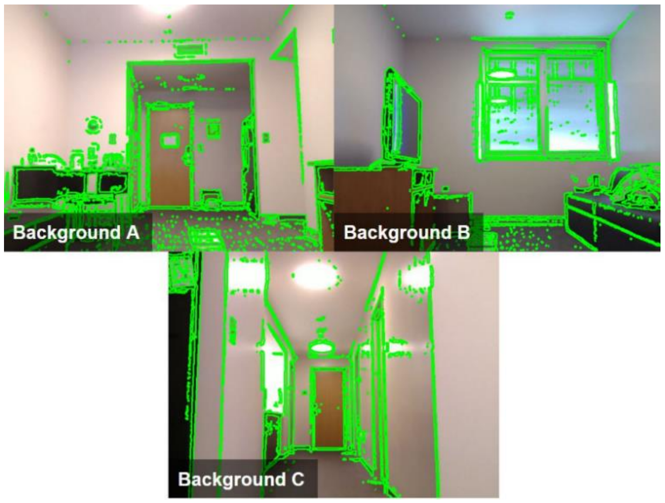

Fig. 8. Noisy backgrounds used in the range evaluation.

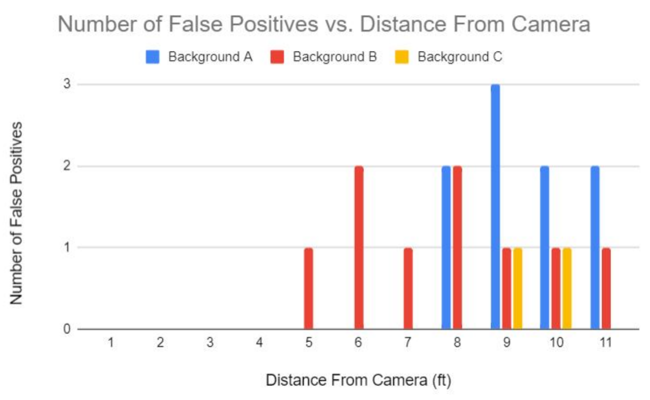

Fig. 9. Number of False Positives vs, Distance From Camera

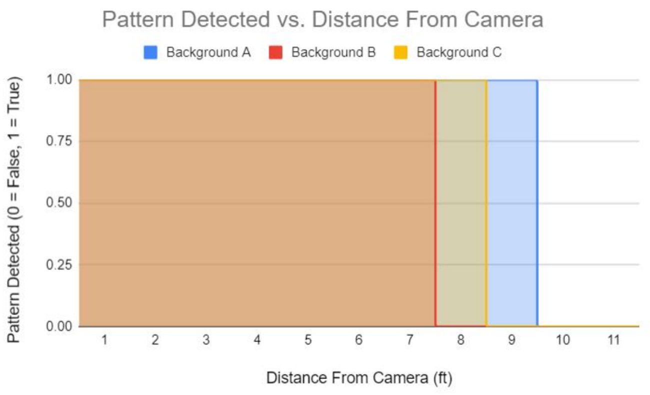

Fig. 10. Pattern Detected (True/False) vs. Distance From Camera

While quantitatively evaluating the range, we also found that the algorithm's performance is dependent on the background. Fig. 11 shows an instance of a constant false positive. The algorithm detects the television's large square shape as a pattern with more confidence than the actual pattern. This is due to the television containing more contours than the actual pattern. Despite this bad detection, the false positive will likely not result in unintended drone commands/behaviors as the resulting decoded bits will not be recognized by the HNN, which is discussed further in the pattern recognition section.

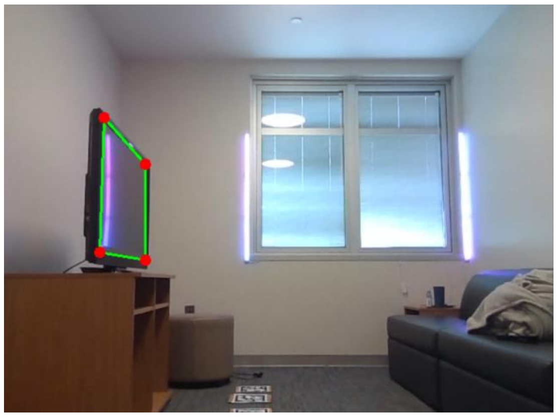

Fig. 11. Instance of false positive detection showing dependency on background.

B. Recognition Results

For testing, the project group iterated each image recognition test twenty times, where each iteration had zero to thirty randomly flipped bits successively. The project group tested with ten patterns. When there were a low number of flipped bits, the HNN's gradient correctly shifted the pattern to the associated pattern in one iteration of the network. Fig. 12 shows that one hundred percent of the patterns are recognized when only five bits are reversed. The accuracy severely deteriorates as more bits are flipped.

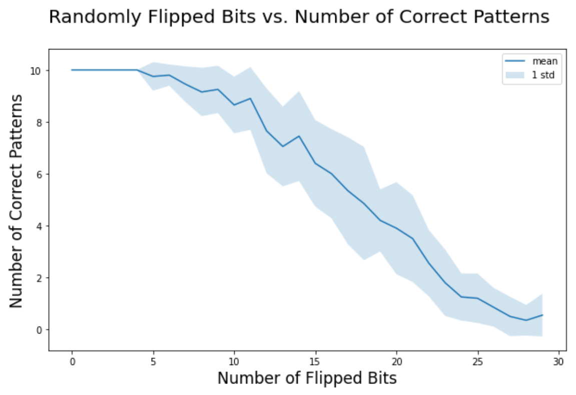

Fig. 12. Randomly Flipped Bits vs. Number of Correct Patterns(20 iterations with 0-30 flipped bits tested)

Unlike the inaccurate single iteration HNN, the HNN with recurrency was much more accurate. When the output was recurrently used as input five times with the same experiment as the single iteration HNN, the pattern recognition accuracy went up to fifteen to twenty flipped bits accurate, as seen in Fig. 13. Furthermore, the network could accurately shift to its associated pattern when the output is input again. However, the caveat to increasing recurrency is increased computational costs, which may not be ideal for real-time autonomous drone flight. There must be research on the optimal rate of recurrency. Nevertheless, the HNN's recurrency feature makes it exceptionally robust.

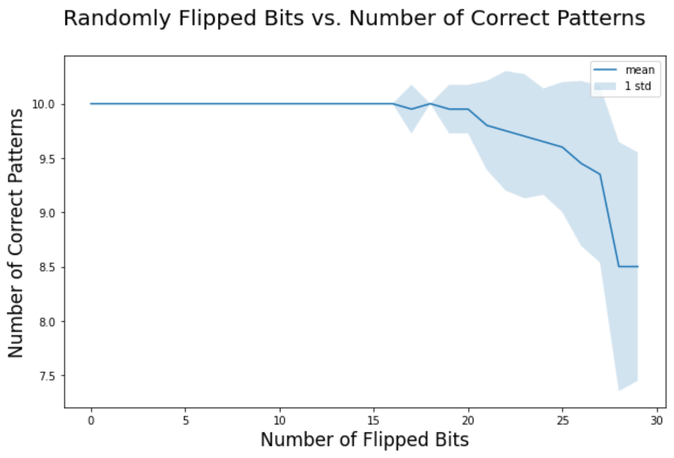

Fig. 13. Randomly Flipped Bits vs. Number of Correct Patterns With Five Iterations Of Inputs Used as Output

V. Discussion, Conclusion, and Future Work

A. Discussion and Conclusion

The pattern detection's lightweight image processing was accurate and robust against differing backgrounds, bright and dark lighting conditions, and various pattern orientations. However, it primarily began failing as the pattern moved further away from the drone's camera. The failure was due to large, dominant background noise and the limitations of using contour area thresholds to remove small noise. However, false positives likely did not cause unintended drone commands/behaviors due to subsequently failed recognition by the HNN. Additionally, the algorithm achieved robust pattern detection at close ranges, which was sufficient for our application.

The use of the HNN enabled a robust pattern recognition algorithm. It could handle up to fifteen to twenty flipped bits, i.e., noise, when running detected patterns through the HNN five times. Each time, the detected pattern was reconstructed closer and closer to its saved pattern equivalent.

The algorithm achieved sufficient, lightweight, accessible pattern detection through OpenCV and robust, relatively computationally inexpensive pattern recognition through HNNs. This ultimately proved that such technologies could be combined to implement a robust, CPU-based computer vision system for autonomous pattern-based drone flight.

The implementation of this algorithm is available at https://github.com/jovanyoshioka/Hopfield-Drone.

B. Future Work

Provided more time, we planned to perform optimization techniques to improve the efficiency of our parameters. The computer vision system would have performed more efficiently if the experiment optimized the number of neurons, patterns, and input runs of the HNN. Our research realized the need for optimization for these parameters as the performance of the HNN was found to be improvable by optimizing these parameters.

First, the project group plans to research the optimal number of neurons for patterns through evolutionary algorithms. The experiment would train an increasing number of patterns to a certain number of neurons to find the optimal number of patterns trainable per neuron. The project group could achieve this using the evolutionary algorithm LEAP, and the fitness function would evaluate the number of patterns that the HNN stably retrieved as more fit.

Second, the project group plans to utilize evolutionary algorithms to optimize the number of times an output must be input again to shift the input pattern towards an associated pattern for the number of noisy bits. The number of HNN iterations can't be too high as the HNN's gradient may result in an unintended pattern to shift towards a recognized pattern. Furthermore, each iteration of the HNN adds to the computational. On the other hand, the project aims to recognize and ignore noisy bits for a robust autonomous drone computer vision system. So, the group plans to implement an evolutionary algorithm to optimize the number of times an output needs to be input again to have the most robust HNN and energy efficiency. The group plans to use the LEAP evolutionary algorithm and the computational cost and accuracy as fitness function parameters.

Third, the optimal number of neurons usable for pattern detection still needs to be optimized. The number of neurons recognized will decrease depending on distance conditions. So, depending on the usage of the drone, the research group would like to optimize the number of neurons in the input pattern for the HNN. Maximizing the number of neurons will provide the most optimal pattern detection with the most optimal number of neurons used for pattern retrieval.

Acknowledgment

This project was created for the Biologically-Inspired Computing course (COSC 420) taught at The University of Tennessee Knoxville in the Min H Kao. Department of Electrical Engineering and Computer Science. Thank you to Dr. Catherine Schuman for her instruction and support throughout the course, enabling this project.

References

[1] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. doi:10.1145/3394486.3406703

[2] "Nvidia clocks world's fastest Bert Training Time and largest transformer based model, paving path for Advanced Conversational AI," NVIDIA Technical Blog, https://developer.nvidia.com/blog/training-bert-with-gpus (accessed May 9, 2023).

[3] D. Krotov and J. Hopfield, "Dense Associative Memory for Pattern Recognition," *Advances in Neural Information Processing Systems*, Jun. 2016.

[4] Jun-Ying Gan and Mengfei Liu, "Face recognition using wavelet packets decomposition and Hopfield neural network," *2009 International Conference on Wavelet Analysis and Pattern Recognition, Baoding*, 2009, pp. 335-339, doi: 10.1109/ICWAPR.2009.5207470.

[5] A. Cyba, H. Szolc and T. Kryjak, "A simple vision-based navigation and control strategy for autonomous drone racing," 2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje, Poland, 2021, pp. 185-190, doi: 10.1109/MMAR49549.2021.9528463.

[6] M. Irfan, S. Dalai, K. Kishore, S. Singh and S. A. Akbar, "Vision-based Guidance and Navigation for Autonomous MAV in Indoor Environment," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-5, doi: 10.1109/ICCCNT49239.2020.9225398.

[7] J. Wang and E. Olson, "AprilTag 2: efficient and robust fiducial detection," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016. doi:10.1109/iros.2016.7759617

[8] Y. Shao et al., "Computer vision-enabled SMART traffic monitoring for sustainable transportation management," *International Conference on Transportation and Development 2022*, 2022. doi:10.1061/9780784484319.004

[9] "About OpenCV," OpenCV, https://opencv.org/about/.

[10] User manual V1, https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20User%20Manual%20v1.4.pdf (accessed May 10, 2023).

[11] N. Soni, N. Singh, A. Kapoor, and E. K. Sharma, "Face recognition using cloud Hopfield neural network," *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2016. doi:10.1109/wispnet.2016.7566167

[12] S.-T. Bow, *Pattern recognition and image preprocessing*, 2002. doi:10.1201/9780203903896

[13] K. Natarajan, T.-H. D. Nguyen, and M. Mete, "Hand gesture controlled drones: an open source library," *2018 1st International Conference on*

Data Intelligence and Security (ICDIS), 2018. doi:10.1109/icdis.2018.00035

[14] F. E. Keddous and A. Nakib, “Optimal CNN–Hopfield network for pattern recognition based on a genetic algorithm,” *Algorithms*, vol. 15, no. 1, p. 11, 2021. doi:10.3390/a15010011

[15] SDK 2.0 user guide, https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf (accessed May 10, 2023).

[16] D. R. Ignatius Moses Setiadi, R. R. Fratama, N. D. Ayu Partiningsih, E. H. Rachmawanto, C. A. Sari and P. N. Andono, "Real-Time Multiple Vehicle Counter using Background Subtraction for Traffic Monitoring System," 2019 International Seminar on Application for Technology of Information and Communication (iSemantic), Semarang, Indonesia, 2019, pp. 1-5, doi: 10.1109/ISEMANTIC.2019.8884277.