UMASS BOSTON
ENGINEERING
DEPARTMENT OF ENGINEERING
COLLEGE OF SCIENCE AND MATHEMATICS

University of Massachusetts Boston
2020 Engineering Expo Proceedings

Real-Time Collision Avoidance Alert System via Deep Learning Image Processing for First-Person-View Drones

Safwa Ali and Huda Irshad

Abstract

This paper presents an open source real-time collision avoidance alert system using deep learning image processing to trigger a microcontroller. The motivation for this project is to integrate the system with first-person-view drone flight controllers to assist drone pilots in avoiding potential collisions. To predict potential collisions in real time, the system inputs real-time velocity vector field calculations into a convolutional neural network, developed and trained on the NVIDIA™ Jetson Nano, a small artificial intelligence computer. A convolutional neural network is a class of deep learning used for image recognition and classification by applying convolution filter layers to learn features in a dataset for future predictions. Vector field calculations are performed by way of optical flow which is the process of detecting and calculating the movement of pixels between consecutive frames. A threshold was set to trigger an alert to the drone flight controller through a universal asynchronous receiver/transmitter, a device for data transmission. The training model achieved 85.6% accuracy with 33.6% loss. When the system was tested, it was able to predict and alert with approximately 72%. Source code and functions for development of the image processing pipeline is found in GitHub [1].

Keywords: Optical Flow, Convolutional Neural Network, Training Model

Safwa Ali, Electrical Engineer, E-mail: *alisafwa11@outlook.com*
Huda Irshad, Electrical Engineer, E-mail: *hudairshad56@yahoo.com*
Client: Martin Luessi and Jennifer Minas, Co-Founders, BrainFPV
Technical Advisor: Dr.Daniel Haehn, Assistant Professor, Computer Science Department, University of Massachusetts Boston

1. Introduction

Machine learning has increased in usage for many computer vision applications, including autonomous vehicles that incorporate collision avoidance. Deep learning is a branch of machine learning that is based on artificial neural networks and representation learning. Large datasets are used in deep learning to develop a model that is trained, evaluated, and validated for making predictions of testing data. In order to incorporate deep learning in a collision avoidance alert system for first-person-view (FPV) drones, this project implements the convolutional neural network deep learning method and the optical flow algorithm on the NVIDIA™ Jetson Nano.

The Jetson Nano [2], a system on module (SOM), is an Artificial Intelligence computer that can run multiple processes in parallel and can accelerate real-time image processing on its graphics processing unit (GPU). This SOM is secondary to the drone's flight controller and works to execute onboard real-time image processing using input video streams from a camera. The system communicates with the flight controller to alert it of any potential crash objects. An alert system for manually controlled mobile devices aids drone pilots in avoiding any compromised paths or potential damages to their drone. By designing and programming the system for integration with the FPV drones, the expected outcome is for drone pilots to improve their flight control. This system is also developed with open source resources and intended to be shared as an open source reference in order to make the contents of the project available to the public for further applications to other projects and/or component features. The image processing aspect of the system uses optical flow and convolutional neural networks (CNN), which are computationally intensive and require modification to function in real-time, with the additional camera as a sensor. A CNN training model involves providing a dataset of labeled images to learn target attributes for further predictions or classifications. However, there is no specific set of target attributes to train for this system because of the different environments in which drones are flown. Consequently, the device takes the vector fields in real-time as inputs to the neural network and trains it to output a prediction of potential crashes within the frame.

2. Background

The functional design of this project is an open source obstacle avoidance system that integrates optical flow with a CNN to detect any potential crash objects and alert the external microcontroller to avoid the object through universal asynchronous receiver/transmitter (UART) communication. As depicted in Figure 1, in the case of this project's specific application, the system is designed with a Raspberry Pi module v2 camera [3] to communication with the NVIDIA Jetson Nano to analyze the real-time video stream frames detected and communicate with BrainFPV [4] drone's flight controller of any potential crash objects. The image processing aspect of the system operates by analyzing the vector field of all objects within the captured frame through an optical flow algorithm and a CNN. The flight controller is then warned once there is a crash prediction from the neural network. The use of open-source optical flow algorithms and neural networks makes it possible to create a system that fulfills this project's functional requirements.

Figure 1. Overall system hardware flow diagram.

2.1 Optical Flow

Optical flow is the apparent motion of objects or surfaces in a visual display, based on the relative change in motion between two consecutive observed moments. In computer vision, an optical flow algorithm produces a 2-dimensional vector field of point movements between consecutive frames of video footage. Mathematically, optical flow uses differential analysis of the same pixels positioned at a small displacement between frames and performs Taylor series approximation to obtain the velocity of the pixels between the two frames. The Lucas-Kanade optical flow method approximates a set of neighboring pixels as a unit, and determines the velocity of the patch of pixels through the same differential analysis. This is only possible with the assumption that the change between consecutive frames is very small and the neighboring pixels in each frame have similar motion. OpenCV, an open-source computer vision library, provides libraries for optical flow that can be utilized for different applications [5]. This project aims to use optical flow for velocity tracking, which is one of its many applications.

2.2 Convolutional Neural Network

A convolutional neural network is a class of deep learning that is mainly used to analyze images for the purpose of image recognition and classification. A neural network is made up of a dataset, layers and neurons. The dataset input to a neural network is trained by applying weight functions to determine the values of the next layer. A convolutional layer in the network simplifies the number of weights for each neuron, convolves the inputs and allows for a quicker neural network process. Figure 2 demonstrates the described architecture of a CNN. Through the training, network convolutional layers, and datasets for training, testing and validation of a neural network, a training model is formed that allows for the identification of trained images.

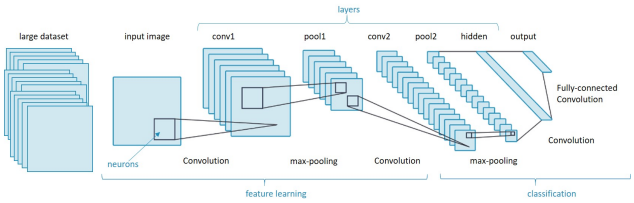

Figure 2. Architecture of convolutional neural network.

In machine learning, the training dataset is used to specify the class labels for the network. TensorFlow™, an open-source machine learning framework, and Keras, a high-level application programming interface which allows for easy use of TensorFlow, are used as references to many training models. One common training model that Keras and TensorFlow provide is the ConvNet model [6], which is a CNN that is trained to recognize handwritten numbers from the MNIST dataset of numerical digits with high accuracy. ConvNet follows a common sequential training model. The image processing system that was designed for this project used a model similar to the ConvNet sequential model but instead of using raw images as our dataset, this model was trained with the target attributes of optical flow vector fields.

3. System Design

The design of this project is based on the goals and the design requirements for the system and specifications of the problem. To meet the goals and requirements of the project the system had to be able to take in a real-time camera stream (placed on a drone or any moving device), process the images to trigger when potential crash objects would be detected, and inform a flight controller of the objects' location in frame, speed and distance. In order to design and create such a system, the following software and hardware specifications were required: An image processing technique implemented in real-time with high accuracy through the use of neural networks; a small high speed digital camera with high resolution compatible with the chosen SOM; and an SOM that is equipped with a camera input port, ethernet port, portable power supply attachment port, and has high speed processing and large (or expansive) memory. An SOM that is compatible to communicate with the BrainFPV drone's microcontroller and the software it uses must be open source.

This section explains the project's design decisions related to hardware, image processing procedure and the integration of all the units in order to fulfill the goals of the design and have a functioning deliverable. Figure 3 depicts an overall diagram of the systems hardware and software pipeline.

Figure 3. System's software pipeline.

3.1 Hardware

The hardware components chosen for this project design are the NVIDIA Jetson Nano as the SOM and the Raspberry Pi 4 module v2 camera for the real-time camera stream input. These two main components were used to build the physical system. The Jetson Nano needs the camera for input, while the image processing and UART communication are programmed on the Jetson Nano, completing the system.

Jetson Nano Environment The Jetson Nano is an SOM that can run high speed modern AI algorithms, making it a small AI computer. It contains connectors and ports for ethernet, microSD card, HDMI output, DisplayPort, DC barrel jack 5V power input, USB, and MIPI CSI camera [2]. The advantage of this SOM is that it can run multiple neural networks in parallel and process several high-resolution sensors simultaneously, which makes it ideal for computer vision and high performance computing. In order to use the Jetson Nano for deep learning image processing, the environment was first assembled with NVIDIA's JetPack and essential packages and libraries for computer vision [7]. Python allows for many open-source libraries to be incorporated, such as OpenCV (which has many resources for real-time applications). TensorFlow and Keras were installed. Jupyter Notebook was also installed and used to simulate the algorithms and neural networks before implementing the processing on the Jetson Nano's GPU.

Camera The Raspberry Pi module v2 camera is a high resolution camera (3280 x 2464 pixels) that can capture approximately 90 frames per second and is compact in size and compatible with the Jetson Nano. It connects through a ribbon cable to a CSI port, which makes it possible to connect to the Jetson Nano's MIPI CSI camera connector port [3]. The Raspberry Pi camera was connected to the Jetson Nano and real-time stream was ensured, using OpenCV source code [8], which was later used for iterative testing of the image processing codes.

3.2 Image Processing

This project's image processing is based on deep learning neural networks to predict potential collisions with images from real-time camera inputs. Due to the various environments that drones are used in, there are no particular objects to train a neural network with, which makes the use of a CNN model alone is insufficient for the purpose of this project. As a result, an optical flow algorithm was implemented on the frames, making the velocity vector field images the dataset for the neural network's training model. The expected result of training the machine with dataset of frames processed with optical flow is that the system can learn to detect potential crashes with the information provided from vector field images. This section discusses the design and development of the image processing pipeline.

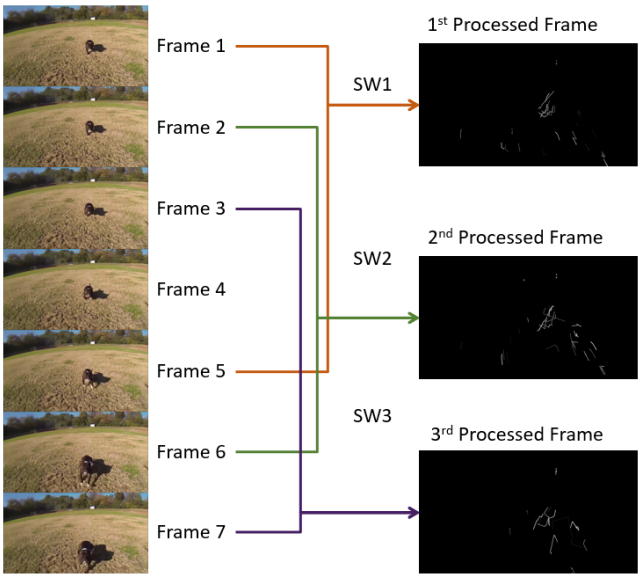

Figure 4. Optical flow with sliding window technique.

3.2.1 Sliding Window Optical Flow Algorithm

The image processing software implemented on the Jetson Nano was developed to calculate the velocity vector field between frames using an OpenCV algorithm for the Lucas-Kanade optical flow method [9]. For the purpose of this project, the algorithm was incorporated in several stages of the design process. Applying the optical flow algorithm to operate in real-time with a camera as the input source pertains to this project because the overall system calls for a real-time camera input. Calculations completed by the algorithm acts as post-processing of the camera capture, extracting information that can not be described from raw images and the sliding window technique was used in conjunction with the optical. The sliding windows technique collects a given number of the most recent frames, and applies optical flow on those frames to calculate the velocity of the pixel movements between the frames. The use of a sliding window technique allows for

the combination of multiple optical flow calculations, making each output image a tracking of vector fields throughout several frames, rather than continuously stacking the previous result from the newly calculated two frames.

As shown in Figure 4, the vector field results for two frames are not continuously being stacked for all frames captured from start to finish by the system. Rather it is doing the processing two frames at a time and stacking the vector field results with results of the next consecutive pair of frames and so on until optical flow is calculated on 5 frames total and that result is a single processed frame. The optical flow images for the training model require preloaded videos as the input. The videos were collected from online drone footage and saved as "jpeg" images to utilize as the dataset. This use of the algorithm allows for the CNN to analyze information that is not attained from the raw camera captures to predict when a crash event will soon occur.

3.2.2 Training Model Dataset

The dataset for this project was generated using Keras' image data generator [10]. The dataset was designed to have two label classes: crash and no crash. Images were organized to make up the training and testing datasets of velocity vector field images collected from the output of the optical flow algorithm. The images for training and testing were modified to gray-scale and pixels were rescaled to accelerate the training of the model. When selecting frames for the dataset, the images in the testing and training sets must be different from each other. The more images in a dataset, the larger the neural network, which increases its accuracy as a result of the improvement in the network's learning. To increase the dataset, the images were augmented.

An important part of the training model is the validation dataset, which is collected from the training datasets and is not trained when compiling the model. Instead, the validation set is used to evaluate the model's fitting through accuracy and loss plots once it is trained. Then adjust the dataset for both, over-fitting and under-fitting training models, by including more images to the dataset to account for the model's accuracy in learning.

3.2.3 Neural Network Design

The CNN model was trained with velocity vector field images. Keras provides a sequential CNN model designed with 3 convolutional blocks for learning and one classification block classifying. After compiling the model, the system was ready to be trained for learning, validating for evaluation, and testing for predictions. As shown in Figure 5, the model was trained and tested on a significant number of epochs, number of iterations the model trains the entire training dataset. Increasing the epochs increases the machine's capability of generalizing the data to adapt to new information.

Label	Training Set	Testing Set
Crash	1612	50
No Crash	1612	50
Total	3224	100

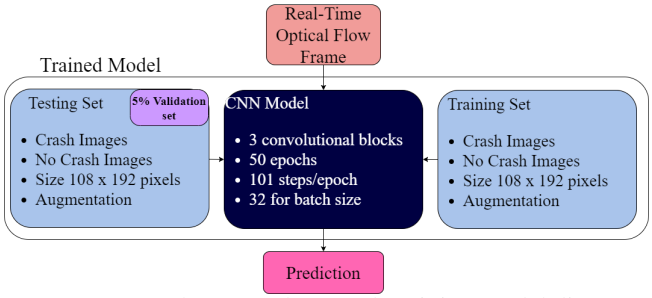

Figure 5. Complete neural network training model diagram.

From evaluations of the model's accuracy and loss for different numbers of epochs the model had been shown to be most effective with a minimum of 50 epochs completed. Since accuracy and loss are two significant factors for the system's success, an adequate training dataset size is required to prevent any over-fitting or under-fitting of the neural network's learning capabilities. To overcome such problems for the neural network, the training dataset was increased by including new images from different footage and applying augmentation, horizontal and vertical flip, a zoom range from 0.8 to 1.2, and a rotation range from -40°to 40°. These parameters affect the training model in accuracy, speed, and fitting of the model from training. The specific values and characteristics of the model are shown in Table 1. This increase in training dataset size from new images and augmentation increases the generalization of the machine to recognize features from new data. The results from providing the machine with a validation set provided information on the network's performance. A validation set was generated by withholding 5% of images from the training dataset and using them to evaluate the model.

3.3 System Integration

The image processing components of this project were merged into one system and implemented on the Jetson Nano's GPU. Figure 3 shows the overall image processing pipeline. A real-time video stream is analyzed through an optical flow algorithm to generate images of the vector field between the frames. The vector field images are then analyzed through CNN to predict if the images are crash or no crash images. As shown in Figure 4, to create the complete CNN, pre-recorded drone frames processed through an optical flow algorithm were used to train a model. To determine if the drone is about to crash onto any objects within the frame, the CNN prediction values are compared to the potential crash threshold value. If there are potential crash objects in the frame, an alert is communicated to the drone flight controller through UART.

Since the training model is predicting crash or no crash in the frame, a boundary has to be determined and set to alert the system before we encounter a frame detected as crash. The solution is using the training model's prediction values of crash and no crash per image to understand the danger level of a crash occurring. When the prediction for crash is the threshold value of 0.3 (meaning prediction of crash is 30% and the frame is labeled as no crash), the SOM is required

to convey the potential crash to the main flight controller. The 30% threshold shows that the camera is not detecting a crash yet, but there are enough features in the image that are similar to that of a crash. Although it has not been tested, with that threshold value, the goal is that the flight controller has enough time to respond to the alert.

As shown in Figure 6, when a potential crash object is detected based on the comparison to the threshold value, a signal is communicated. For testing, a phrase was printed as the communication signal. The code for UART was written to transmit the signal that is decoded as "crash" and then close immediately. UART is the final form of communicating the signal.

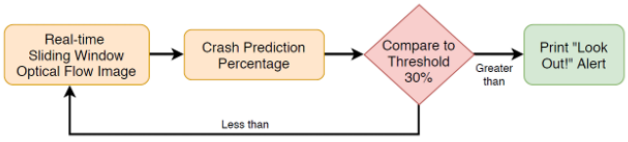

Figure 6. Prediction alert threshold and UART communication.

4. Results

The results of this project were produced through analysis of the neural network's training model, the predictions of the real time sliding window optical flow images, and the accuracy of the complete system's predictions.

4.1 Optical Flow Results

Optical flow has shown to be helpful for a processing technique as a form of extracting data for the neural network's analysis and learning. The optical flow results from the preloaded frames had indicated varying signs of potential crashes. However, the optical flow algorithm has also shown some disadvantages. Mainly, the algorithm assumes small motion between frames which is not a characteristic that can be guaranteed in every situation for drone pilots. It can also lead to misinterpreting pixels that have left the frame to pixels going in the direction of new pixels that have just entered or of the same gray-scale value. This error could escalate over time if the sliding window technique was not used. The sliding window prevents this escalating error by limiting the results to be calculated on 5 frames and the technique slides over one frame for the next calculation. As a result from calculations slowly transitioning to new frames, future calculations will not be affected by the frames with this miscalculation.

4.2 Neural Network Results

With the configured CNN model explained in Section 3.2, the dataset for the training model was configured and the neural network was trained. Figure 7 shows the difference between the crash and no crash class images in the dataset used to train the model. When there is no crash, the vector field flow is continuous linear or parabolic lines. This is because there is no sudden change between frames. However, with the crash labeled images, there are many jagged lines that show sudden changes between the frames, representing a potential obstacle in the camera's path. The result of the 5 real-time frames is an optical flow image illustrating the vectors of motion between the frames. Lastly, all images were modified to match the training model's image size and run through the training model to predict the likelihood of the image is crash or no crash.

Figure 7. Images from dataset labeled no crash (left) and crash (right).

When we initially trained the model with a much smaller dataset, over-fitting occurred in the model. To improve the neural network, the training model was modified and improved in several ways. The most important change is increasing the dataset size. We were able to create a dataset of approximately 3300 images. It was important to increase the epochs from 4 to 50. The current model was trained with 50 epochs. As shown in Figure 8, this resulted in a complete model with approximately 85.61% accuracy and a loss of approximately 33.61%.This is an improved accuracy level, however the loss of data remains high. If given more time, it would have been best to improve this model by increasing the dataset size (ensuring that the images in each class are accurate) and training the model with more epochs and larger convolutional layer filter sizes.

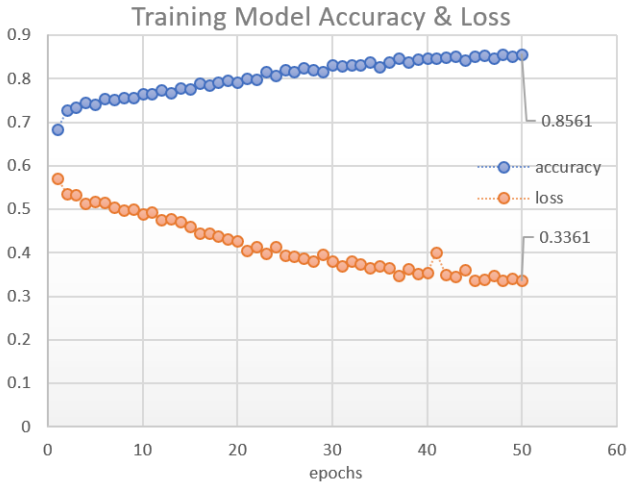

Figure 8. Accuracy and loss plot of training model.

4.3 Implementation: Pipeline and Hardware Testing

In order to test the accuracy of the image processing pipeline's execution for predicting crash or no crash labels for real-time frames and potential crash threshold, the Raspberry Pi camera

was connected to the Jetson Nano and a continuous live stream of the optical flow frames was displayed to allow for real-time visual tracking during the testing. A sample of the resulting display is shown in Figure 9.

Figure 9. Real-time prediction and alert for optical flow images.

When the complete pipeline software was programmed on the Jetson Nano (with the real-time camera input, optical flow image processing using CNN and UART), it was tested in a stationary setup because in order to detect if a signal was being transmitted via UART, a logic analyzer was connected to the system and observed using the Saleae^{TM} Logic Analyzer Software, as shown in Figure 10. The UART will close after writing the signal, at a baud rate of 115200 for 8 bits and no parity, and one stop bit. The figure below displays the implementation of the UART and its setup. It can be viewed that the Logic software was able to detect the transmitted signal and decode it as "crash" from the Jetson Nano. Improvements that could be made for the UART are to include parity, start bits, communicate the predicted direction of the collision, and have the system wait for confirmation from the flight controller that it has received the signal before closing. The reason these modifications were left out for this project is because of the concern of timing on waiting for a response because it would impact the speed in which we can calculate real-time optical flow.

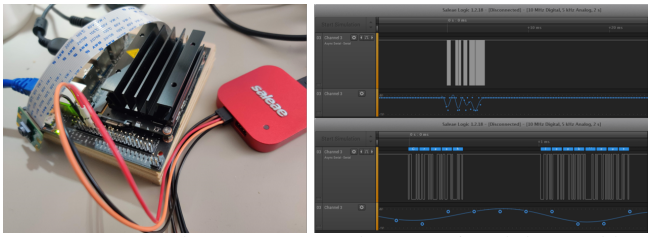

Figure 10. Hardware system and UART signal detected by logic analyzer.

As a result of testing the complete system and image processing pipeline with the trained neural network, it was found that the system only predicts 72.8% of the potential crashes in the camera's path. Table 2 shows the results of 5 tests and the amount of crash predictions made during each test with the percent of correct predictions.

Test No.	No. of Predictions	Accuracy %
1	25	60
2	25	92
3	25	84
4	25	56
5	25	72

5. Conclusion

The deliverable for designing the real-time collision avoidance alert system were completed using the Raspberry Pi v2 camera interfacing with the NVIDIA Jetson Nano and processing the images to input into the training model for predictions to send an alert. Optical flow calculations was used as a form of processing for the raw images and incorporated the sliding window technique, with a size of 5 frames, which resolved the optical flow miscalculation for large motions from impacting future calculations. The CNN model was designed with three convolutional blocks and one classification block and trained for 50 epochs with vector field images of 1600 for each label, resulting in a performance of 85.61% accuracy and 33.61% loss. The model performance can be improved further with an increase in the dataset and number of epochs for training. All resources used and code developed is open source and is accessible on the GitHub [1] because the deliverable, even though designed for first-person-view drones, is versatile and can be applied to many other projects.

References

[1] S. Ali and H. Irshad. Complete project scripts. [Online]. Available: https://github.com/safwali11/UMBENGIN492.git

[2] NVIDIA. Jetson nano — nvidia developer. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano

[3] R. P. Foundation. Camera module v2 raspberry pi. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Raspberry%20Pi%20PDFs/913-2664_WEB.pdf

[4] BrainFPV. Brainfpv home. [Online]. Available: https://www.brainfpv.com/

[5] Opencv. [Online]. Available: https://opencv.org/

[6] Mnist cnn - keras documentation. [Online]. Available: https://keras.io/examples/mnist_cnn/

[7] A. Rosebrock. Getting started with the nvidia jetson nano. [Online]. Available: https://www.pyimagesearch.com/2019/05/06/getting-started-with-the-nvidia-jetson-nano/

[8] OpenCV. Getting started with videos — opencv-python tutorials 1 documentation. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

[9] ——. Optical flow - opencv-python tutorials 1 documentation. [Online]. Available: https://keras.io/preprocessing/image/

[10] Keras. Image preprocessing - keras documentation. [Online]. Available: https://keras.io/preprocessing/image/