




High Speed Marker Tracking for Flight Tests

Gabriel A. Melo , Marcos R. O. A. Máximo  and Paulo A. L. Castro 

Abstract—Flight testing is a mandatory process to ensure safety during normal operations and to evaluate an aircraft during its certification phase. As a test flight may be a high-risk activity that may result in loss of the aircraft or even loss of life, simulation models and real-time monitoring systems are crucial to access the risk and to increase situational awareness and safety. We propose a new detecting and tracking model based on CNN, that uses fiducial markers, called HSMT4FT. It is one of the main components of the Optical Trajectory System (SisTrO) which is responsible for detecting and tracking fiducial markers in external stores, in pylons, and in the wings of an aircraft during Flight Tests. HSMT4FT is a real-time processing model that is used to measure the trajectory in a store separation test and even to assess vibrations and wing deflections. Despite the fact that there are several libraries providing rule-based approaches for detecting predefined markers, this work contributes by developing and evaluating three convolutional neural network (CNN) models for detecting and localizing fiducial markers. We also compared classical methods for corner detection implemented in the OpenCV library and the neural network model executed in the OpenVINO environment. Both the execution time and the precision/accuracy of those methodologies were evaluated. One of the CNN models achieved the highest throughput, smaller RMSE, and highest F1 score among tested and benchmark models. The best model is fast enough to enable real-time applications in embedded systems and will be used for real detecting and tracking in real Flight Tests in the future.

Index Terms—tracking, convolutional neural networks, corner detection, deep learning, flight test, fiducial marker, Secchi disk.

I. INTRODUCTION

This paper provides a fast and precise neural network model to detect fiducial markers in real-time embedded systems, namely, the Secchi Disk fiducial marker. Those are used in many applications which is necessary to track an object in real-time, such as flight testing, automobile testing, mixed reality, optical experiments, and motion capturing for robotics or animation. In special, the flight test application requires certified hardware for airborne operation and optimized software to run in real-time as depicted in Fig. 1 which demonstrates a store separation flight test. These fiducial markers, as they have a predefined shape with fixed and distinct characteristics from their environment, such as high contrast edges, they are easier to detect than the arbitrary object that they are attached to. Those applications often need high precision and low misclassification, thus an approach using the whole object would encompass a high computational cost. Therefore, tracking a fixed amount of fiducial markers distributed along



Fig. 1. A store separation flight test with an Embraer 314 (Super Tucano) conducting the test with a camera pod attached (middle) and its image captured before the separation (lower).

with the whole object also keeps the computational burden manageable while also increasing the tracking size for a rigid body, resulting in better accuracy.

In-flight processing also allows for multiple separations to be safely performed in the same flight, reducing the overall flight time in an airworthiness certification campaign, therefore fuel consumption and costs [1]. Being able to process the data in real-time also increases the overall safety of the mission, as opposed to processing the data on the ground, after the flight, as the telemetry link is limited and not capable of transmitting high-speed and high-resolution images. These are the necessities that came from the Brazilian Flight Test and Research Institute IPEV (*Instituto de Pesquisas e Ensaios em Voo*), an organization focused on guaranteeing the safety and airworthiness of the Brazilian Air Force's aircraft by performing flight tests.

Other markers such as ArUco (Augmented Reality from University of Cordoba) and ChArUco (Chessboard ArUco) [2] are more commonly used in mixed reality applications, as they provide a more flexible setup and the capacity to uniquely identify each marker. For more dynamic applications, even natural landmarks may be used as a target for tracking, requiring either a multi-camera setup or knowledge of the world to estimate the target's size [3], though the computational power required to process it in real-time at a high rate would not be possible to be embedded.

As related works, several classical algorithms are used for identifying many markers composed of crosses, circles, edges, lines, edges, and corners, such as the: Harris Corner detector, Shi-Tomasi Corner detector, SIFT (Scale Invariant Feature Transform), SURF (Speeded-up Robust Features),

Gabriel A. Melo is with Instituto de Pesquisas e Ensaios em Voo (Flight Test and Research Institute), São José dos Campos – SP – Brazil gam@ita.br
 Marcos R. O. A. Máximo and Paulo A. L. Castro with the Autonomous Computational Systems Lab (LAB-SCA), Computer Science Division, Aeronautics Institute of Technology, São José dos Campos – SP – Brazil
mmaximo@ita.br, pauloac@ita.br

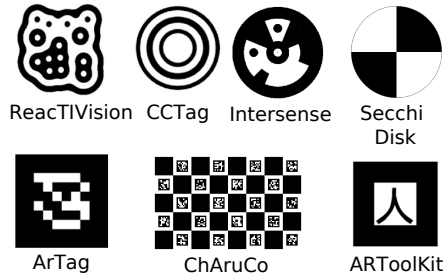


Fig. 2. Various fiducial markers from the literature were created for different applications and augmented reality libraries. Adapted from [8] and [9].

FAST (Features from Accelerated Augmented Test), BRIEF (Binary Robust Independent Elementary Features), and ORB (Oriented FAST Rotated BRIEF) [4].

In comparison to those handcrafted approaches for marker identification, a machine learning solution offers more flexibility to detect different shapes and colors of markers as its adaptation would only require changes to the training set and not to the code itself that would happen, for instance, if instead of an edge in the center of the marker another circle were to be used. Among the learning-based techniques, a convolutional neural network is shown to have the best inference time, and, depending on its training set, a generalization capacity at least equal to those classical feature-based approaches [5]. Recent works, such as Deep Charuco [2], also employs CNNs to Charuco markers detection. Other markers used mostly in mixed reality applications are shown in Fig. 2. To the best of our knowledge, this is the first work to use CNNs to detect Secchi disk fiducial markers in a flight test application.

After determining the bidimensional coordinates of known points in space and after estimating the intrinsic camera parameters, one may compute the tridimensional position and orientation of given points with respect to the camera. This problem is known as Perspective-of-N-Points (PnP) [6] and it serves as a source for feedback information regarding the markers' localization. For a multi-camera setup, another approach is to triangulate the point's position given the coordinates of one camera with respect to the other [7].

In this work, Section II covers a summarized overview of the deep learning theory, Section III describes the methodology used for data generation, model implementation, and training, Section IV presents the results and discussions and the conclusions are presented in Section V.

II. DEEP LEARNING THEORY

Artificial neural networks are arbitrary computational graphs whose inspiration came from its biological counterpart [10]. Deep learning bootstraps this idea by envisioning several techniques for enabling the training of more stacked layers of neurons, creating computational graphs with greater maximum node distances [11].

Convolutional neural networks (CNNs) are essentially a set of kernels of cross-correlations that are applied in parallel (in the case the kernels pertain to the same layers) or sequentially. Those kernels have differentiable activation functions and their parameters are learned through back-propagation [12].

Therefore, the synaptic weights are expressed by the elements from those convolution kernels. Compared to a fully connected layer, the convolutional one is a sparser representation as only nearby (on pixel location) neurons are connected. It requires fewer parameters for the computation as the same kernel is used throughout the image (weight-sharing) and also preserves the dimensional structure of an image as its input and outputs are channels of matrices [13].

The convolution operation is actually a direct cross-correlation from the kernel to the input image. To interpret this operation by the mathematical definition of convolution, one needs to interpret the kernel's representation as being a kernel mirrored in both x and y axis. It is mathematically defined as follows, where C is the number of channels, KH is the kernel's height, KW is the kernel's width, a_c is the activation value at the channel c , and k_c is the c -th kernel, resulting in the activation of the z_{ij} neuron:

$$z_{ij} = \sum_{k_0=0}^C \sum_{k_1=0}^{KH} \sum_{k_2=0}^{KW} k_c(k_1, k_2) a_c(i + k_1, j + k_2). \quad (1)$$

For more details regarding Deep Learning and CNNs, please refer to [14].

III. METHODOLOGY

As a fiducial marker, the Secchi Disk was used for historical reasons, as most recorded videos from automobile and flight testing used either a black and yellow or white and black edge encompassed inside a circle, as observed in the store under the wing in Fig. 1.

The tracking process was done in a frame-by-frame application of the detection method, using the nearest neighbor algorithm to match associate each marker from its previous location to a later frame. As such the previous location defines a region of interest (ROI) in which the marker should be located in the next frame. The reprojected points from the 6D estimates given by the PnP algorithm were also utilized as a reference for defining the ROIs. In case of divergence, the whole image would be scanned for detected markers.

A. Synthetic Data

The training dataset for the learning algorithms was modeled as a 2D SVG (Scalable Vector Graphics) figure whose properties were randomized: colors, position, rotation, X and Y scale, and rotation after scale. Those values were sampled from a uniform distribution with 32-bits floating points numbers, the position had 4-pixel padding from the image's borders, the radius varied between 8 to 24 pixels, and the colors, for each channel, had the darkest value sampled from 0 to 80 and the other colors were the sum of this value with another sampled from 20 to 60 and 60 to 150 corresponding to the background and brightest color respectively. A Python function wrote a set of randomized SVG codes which were rendered to PNG using the CairoSVG library.

Initially, the SVG figure was composed of a rectangle that comprised the background and a group that was formed from four quarters of a circle. A prior approach with a full circle and

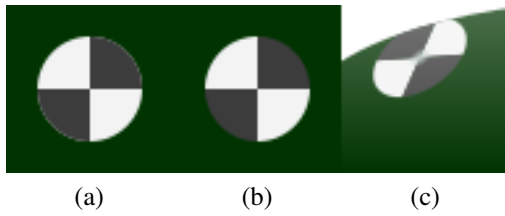


Fig. 3. Artifact in the border of the marker (a), image rendered without the artifacts (b), and image rendered with light and curvature emulation (c).

two quarters resulted in aliasing artifacts in the border in which the full circle color could form a one-pixel circumference involving the mark as shown in Fig. 3.

Further iterations from this dataset generator added a variable background in an elliptical shape that would try to emulate a marker placed on a curved surface. Color gradients were also added, not only in the background, emulating lights from different directions but also a circular bloom effect centered on the marker, a simple attempt to recreate the glossiness of the sticker.

During training, several image augmentation techniques were also employed such as adding random Gaussian noise, random Gaussian blur, salt, and pepper noise, and even a Bayer color filter array was employed in a separate model that would be fed raw images from a color camera. Those noises and augmentations were applied over a model that was first trained on the easier (initial) images in order to facilitate the learning task.

The data generator allows for an arbitrary number of markers to be generated as there are several degrees of freedom in real-valued parameters. For the simpler models, the generated images had a resolution of 24 pixels wide square, but resolutions of 64 pixels were also used in the mixed model. Over 320,000 images were rendered of which 16,000 (5%) were used for validation (hyper-parameter selection and early stopping) and 16,000 (5%) for testing (final model evaluation). The main limitation for not generating more images was so that the entire dataset could fit in the shared GPU memory. For the detection, the datasets were doubled in size with negative samples.

Even though the rendered images are colored (3-channel RGB), they are either converted to gray-scale directly or to a Bayer pattern image, both a 1-channel representation that would be the raw output of respectively a mono or color camera. This makes the task somewhat more difficult, as information from other channels is lost. Even though it would be possible to use linear interpolation for demosaicking prior to feeding the neural network, several neural models for demosaicking and super-resolution have been developed [15]. For the first trained models and for comparisons purposes with OpenCV's algorithms, the grayscale direct conversion was used, according to the following Equation, where R , G , and B represent the red, green, and blue channel values for each pixel:

$$grayscale = 0.299R + 0.587G + 0.114B. \quad (2)$$

B. Model Implementation

The task in which the implemented models aimed to solve was to estimate the fiducial marker's position with subpixel accuracy (its x and y coordinates) while also providing a measure of confidence, the probability that the image was that of a trackable marker. While restricting the problem to tracking a single marker would seem to oversimplify the main problem which requires multiple markers on an object of interest, there are some techniques that allow expanding those single estimation models to multiple trackers [16].

The first model implemented was a fully-convolutional network with five convolutional layers, using a ReLU activation function. The input image was a single-channel square grayscale image 24 pixels wide. All the layers had a square kernel of 5 pixels in size, except the last layer which had a size of 6 pixels. The number of channels in each layer exponentially increased until the middle layer (by a factor of two), and then decreased in the same manner in order to match the output dimension of 2, as shown on the model's diagram in Fig. 4.

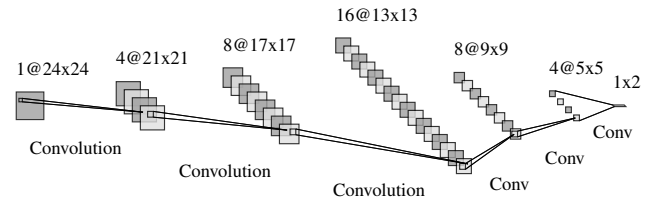


Fig. 4. Architecture representation for the first iteration of the fully convolutional model, with x and y estimates as outputs.

Another model which tried to explore the equivariance property from the convolutional layers was also developed. Its first five layers were also fully convolutional with a kernel of size 5 but it utilized a zero-padding on the image to preserve its width and height on those layers while increasing the number of channels. The centroid from the activation of each channel in the last convolutional layer was calculated and served as input to a fully connected sequence of layers whose final output was the marker's position and confidence. One implementation of this model using Bayer pattern [15] raw images of markers featured a downsampling operation followed by an upsampling operation, which is performed by the first convolutional layer that had a stride of 2 and by a bilinear interpolation respectively. This was thought to separate and aggregate the colors in 4 channels with half the resolution which would be combined to debayer the raw image, as represented in Fig. 5. This encourages the network to learn a fast demosaicking method specialized for the localization problem that should at least outperform a single channel-wise linear interpolation demosaicking.

Besides the Xavier Initialization [17] which is random sampling with a variance proportional to the fan-in of the layer, the weights of the first layer for this mixed model were also summed to canonical kernels that would facilitate the training. Kernels with Bayer filters, Sobel filters, and the identity kernel were employed in this transfer learning-like technique, each

applied on its own channel in the first, second, and third convolutional layers respectively.

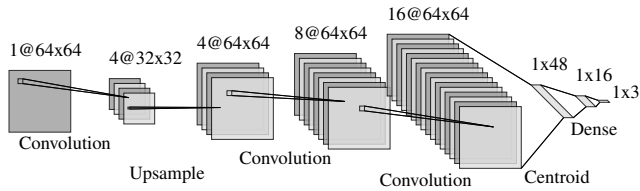


Fig. 5. A mixed convolutional model that employs convolutional layers maintaining the image dimension while increasing the number of channels while calculating the sum, x and y softmax positions for the activations of each channel and passing to fully connected layers.

Besides an initial usage of MSE (mean squared error) for all normalized outputs (x and y between 0 and 1 and confidence), the loss function was defined as a mixture of BCE (binary cross-entropy) and MSE, as shown in Equation (3), masking the latter when the target probability was zero. This way when the input image did not represent a marker, any positional output given by the model would not influence the loss, thus becoming *don't-care* values. The P variable indicates the true probability of the image being a marker (and was restricted to either 0 or 1), \hat{P} was the model's probability. The same applies to the x and y parameters which represents the normalized positions. The K hyper-parameter determines the relative importance between the task of estimating the position in relation to the probability of the image being a marker. For all trained models, this constant K was chosen as the area of the image in pixels, as it denormalizes the position and thus an error of 1 pixel would have the same order of magnitude as the BCE of a random guesser.

$$L = \frac{1}{N} \sum_{i=1}^N \left((\hat{x} - x)^2 PK + BCE(\hat{P}, P) \right), \quad (3)$$

where the BCE (Binary cross-entropy) is defined as follows:

$$BCE(\hat{P}, P) = -P \log(\hat{P}) - (1 - P) \log(1 - \hat{P}). \quad (4)$$

For training, a normalization was used both for the inputs and for the outputs, constraining it to values between 0 and 1. Therefore, both x and y were linearly transformed from pixels units to relative image size units, with (0, 0) representing the topmost left position and (1, 1) representing the rightmost bottom position. As for the image input, the unsigned integer 8-bit values representing each pixel intensity were converted to floating-point numbers linearly with 255 mapped to 1.0.

For the models that estimated the probability that the image contains a marker, negative samples extracted from background images from different scenarios were used. Random crops were randomly extracted from those images so that there would be the same number of positive and negative samples. In addition to those, constant images were also used as negative samples.

The models were trained for up to 300 epochs but generally would finish training before 100 epochs due to the early stopping heuristic, as observed in Fig. 8, with a patience threshold of 10 epochs. This measure, in addition to the noise, data augmentation, and variations present on the training data

as well as the limited number of parameters in the networks, mitigated over-fitting from occurring, at least given that the training, validation, and testing sets were sampled from the same distribution. The training code was written in Python 3.9 using the PyTorch 1.8.1 library [18] and executed on one node in the Santos Dumont (SDumont) supercomputer cluster.

C. Multi-Task Learning

For the last model, which relied on calculating the centroid of the neuron's activations, a multi-task learning setup was devised which consisted of learning a marker position heatmap. A Gaussian distribution with a standard deviation of 2 pixels and a mean equal to the marker's center position was employed as the main feature of this heatmap. Thus, the model had an extra output that reconstructed the heatmap, as shown in Fig. 6.

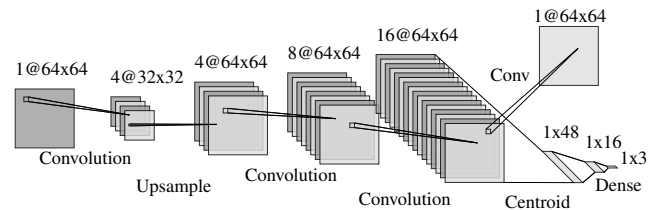


Fig. 6. Multi-task learning adaptation for the mixed model as the last convolutional layer is used as input to reconstruct a heatmap whose highest-value point is the marker center.

D. Progressive Training

One technique that made training converge in fewer epochs than just feeding the most elaborate rendering with blur, bloom, and noise was to first feed the simpler images and then proceed to gradually add those effects. The crisp corner on the center of the marker was the most prominent feature that the models identified, as a result, obstructing, blurring it, or applying a bloom effect over it would severely reduce the accuracy of a newly trained model from scratch.

Another trick, easily applied in the case of the same padding (zero-padded images) was to gradually increase the number of convolutional layers, creating each layer at a time after some training of the already existing ones. For initializing a newly inserted layer inside an already-trained network, its weights were adjusted so that the expected mean values from the activations of an earlier layer would remain the same and its variance would not change more than an order of magnitude, typically increasing by two-fold. For a square kernel of size 5, it would mean to randomly initialize its weights with mean zero and variance 1/5 sampled from a normal distribution, except for its center weight which would be set to 1.

E. Multi-Tracking Generalization

An initial approach for detecting and tracking multiple markers from a larger image would be simply to cut several smaller windows forming a grid (preferably with some overlapping so that a marker placed exactly on the limits of

a window would fall in the range of another) and run the inference on each of those smaller images separately.

A better approach for efficiently leveraging the computation performed by the convolutional layers is to input the whole image to the convolutional layers, resulting in a higher-dimensional output by those layers that are addressed by a larger stride in the last layer. For the fully convolutional model, this is implemented by defining a stride and by applying a non-maximal suppression to the last layer, as depicted in Fig. 7. Note that the stride is applied in the last convolutional layer and implicitly defines a square bounding box for each activation, whose size is the same as the original single input model (24x24).

An interesting result of this rationale for the model with flattened fully connected layers is to convert those layers to convolutional layers with kernels of size 1, which is mathematically equivalent. The number of neurons in a layer becomes the number of channels and its weights remain the same. In this way, the latter method may be applied as the network became a fully convolutional one.

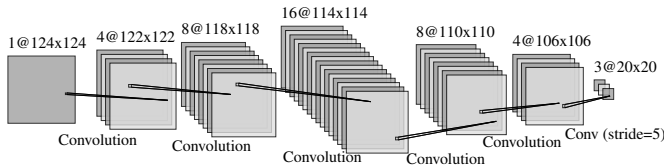


Fig. 7. Resulting architecture for applying the same convolutional model to bigger images for multi-tracking.

IV. RESULTS AND DISCUSSIONS

The models were evaluated on an Intel Core i7-5500U CPU @ 2.40GHz notebook which also had an integrated graphics processing unit (Intel Gen8 HD Graphics) which was used in the cases that would result in a better performance. The computer had 8GB of dual-channel DDR3 @ 1600 MHz, and run in an environment with Python 3.8.6, Linux 5.10, OpenVINO™ build 2021.3.0 API 2.1. Such a system is similar to the airborne certified computer inside the camera pod and it is also representative of single-board computers (SBCs) that are typically available in edge devices for embedded applications.

For the OpenVINO™ compilation, the models were ported to an ONNX (Open Neural Network Exchange) format and then ran in the inference engine. More optimizations for inference could be performed as the model kept all of its parameters and computed in the same floating-point precision of 32 bits. The code developed in this work is openly available in the following Github repository: <https://github.com/gabruip/papers>. While the original store separation videos are not provided in the repository, synthetic data and other captured images are available.

As the upsampling operation is not supported in the ONNX format and its OpenVINO™ implementation, the mixed model was retrained without the downsampling and upsampling operations and also had its centroid operation changed to a fixed convolutional kernel.

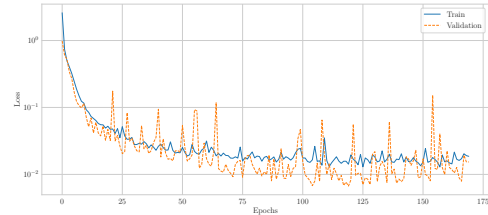


Fig. 8. Training and validation error measure in the loss function defined by Equation 3 for the normalized values in the fully convolutional model.

TABLE I
RESULTS FOR INFERENCE USING THE TRAINED MODELS

| Model | RMSE | F_1 | Acc. | FPS |
|------------------------------------|-------|-------|------|--------|
| Shi-Tomasi | 0.75 | 0.87 | 0.88 | 39,525 |
| Harris | 1.06 | 0.79 | 0.82 | 38,022 |
| SIFT | 2.72 | 0.78 | 0.81 | 1,167 |
| OpenCV's <code>cornerSubPix</code> | 0.37 | 0.86 | 0.84 | 24,630 |
| HSMT4FT | 0.086 | 0.99 | 0.99 | 47,944 |
| Mixed Centroid | 0.11 | 0.99 | 0.99 | 19,679 |
| HSMT4FT Wider | 0.067 | 0.99 | 0.99 | 5,252 |

A. Comparisons

The OpenCV library [19] was used since it has an optimized implementation for Intel processors. The function `cornerSubPix` was employed as it would have better accuracy for detecting the quadrangular edge with an error less than one pixel, developed by Forstner [20], which was verified with a root mean square error (RMSE) of 0.37 for the simple synthetic markers, as shown in Table I, which also includes the F_1 score of the detection rate (defined as the harmonic mean between recall and precision). Other algorithms such as Harris and Shi-Tomasi had a higher throughput but they did not reach a sub-pixel accuracy. The HSMT4FT model achieved the highest throughput (47,944) and an error (0.086) which are 95% higher and 76% lower, respectively than those achieved by `cornerSubPix`. There is a trade-off between the model size and its accuracy, as observed in the results from the mixed and the wider models which achieved higher accuracy at the cost of lower throughput. All the neural models achieved over 99% classification accuracy in differentiating marker images from background images in the test set, considering a 0.5 threshold in the probability.

A qualitative comparison for the tracking can be observed in Fig. 9 in which the proposed method (upper) is compared to the classical one (bottom). While several markers were lost or not even tracked in the older method, the newer was able to track and estimate the position of every marker visible during the store separation. Those store separation flight tests were executed previously by IPEV with an older algorithm implemented in MATLAB® that could take up to 4 minutes to process the video stream. The data recorded from those previous flights test were used to evaluate the newly proposed method, which is expected to be executed in flight during the next mission.

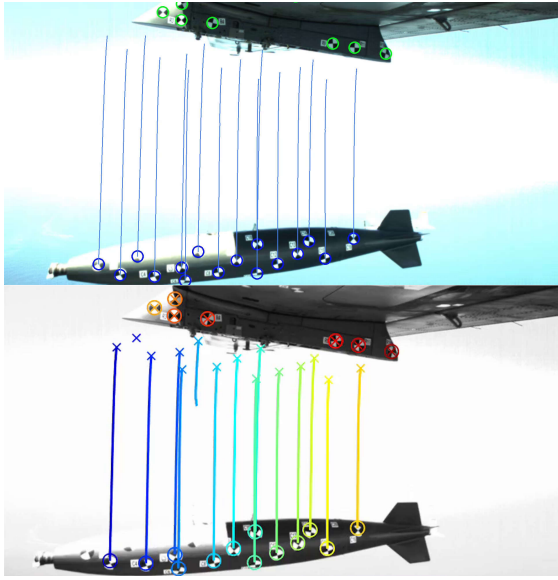


Fig. 9. Comparison between the tracking between the proposed novel model (top) and the classical edge detection approach (bottom), which was unable to track three markers on this launch. The color differs between the methods because of the debayering step used on the HSMT4FT model. Video comparison available in <https://youtu.be/etf37r0CQA0?t=195>.

B. Ablations

Notice that for the Initial Fully Convolutional model, an even simplified version that had no measure of confidence was initially trained, as a direct comparison to OpenCV's `cornerSubPix` function that only provided the x and y coordinates as outputs, as shown in Fig. 10.

The Leaky ReLU activation function had a similar performance to its ReLU counterpart. Nevertheless, there were instances when training the ReLU models in which most of the neurons had their activations permanently set to zero because of negative weights, so the network had to be reinitialized from scratch. This was typically detected in the first 10 epochs as both the training and the validation losses would stay approximately constant at a high value (that indicated that sub-pixel accuracy had not been reached).

Another test was to train a shallower model. With the fully convolutional model, this would be achieved with larger kernels and/or with a kernel stride larger than one. Those shallower models would have a higher throughput but at the cost of lower precision.

An interesting note was that the engineered approach to calculate the centroid and sum of the activation did not outperform the fully convolutional models when controlling for both the number of learnable parameters, precision, and throughput. The computation for the centroid is mathematically equivalent to the convolution with a kernel that is a ramp function on the desired axis, normalized by the activation.

The downsampling and upsampling operation for the mixed model increased the error for the grayscale images when compared to its counterpart, as those operations suit better to Bayer pattern images. Removing channels in the middle layers of the model also resulted in a loss in performance, albeit not as large as if the same number of parameters were

to be removed from the initial or last layers which already had fewer channels than those middle ones. An exception to this observation was the model whose

The NAdam gradient update heuristic was actually very similar to its Adam counterpart, though in some iterations, mainly with ReLU activation functions, the error diverged when employing the NAdam implementation in PyTorch. The other time constants related hyper-parameters β_1 and β_2 should also be tweaked to reduce this erratic behavior caused by larger gradients. Nevertheless, the hyper-parameters which had the biggest influence on the convergence were the starting learning rate and its decay rate.

The Gaussian's blur on the image significantly increases the position error, although not as large as the variance of the Gaussian kernel, which suggests that even though the networks have learned to detect the sharp quadrangular edge location, it also utilize the information from the lines that should converge at the edge center.

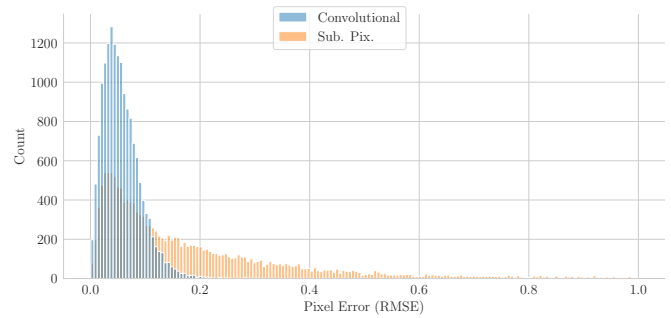


Fig. 10. The positional error distribution for the test dataset compared between the fully convolutional model and OpenCV's `cornerSubPix` corner detection.

V. CONCLUSION

Several convolutional neural network models were developed and trained to track fiducial markers from a high-speed video in an airborne embedded system. Those models were compared with several other corner detection algorithms, such as the Harris Corner detector, the Shi-Tomasi Corner detector, and other features detection, such as SIFT, and were found to have a better detection rate and a lower localization error.

While not every convolutional model implemented could surpass its OpenCV counterpart in terms of inference throughput (FPS), as it requires more arithmetic operations per second (such as FLOPS), their accuracy metrics were better by at least a factor of 3. Nevertheless, wider models were able to have greater precision than their counterparts with the same number of layers. It seems to indicate that there is a trade-off between precision and inference throughput for the same computational device. The Fully Conv. Initial model achieved the highest throughput, smaller RMSE and highest F1 score among tested and benchmark models as presented in section IV-A. This model is suitable for real-time applications in embedded systems and will be used for real detecting and tracking in real flight tests in the future.

In future works, the developed models could also be applied for ArOuco and ChArOuco markers detection, requiring

only adjustments to its training set in order to incorporate square edges on the task of sub-pixel position refinement. The same expansion techniques would be applied for whole image detection. Another important future work is to evaluate and improve the model's generalization capabilities in other datasets that encompass other markers in different conditions, and also comparing it to other deep neural network models. Pruning the networks or reducing the computation precision of some initial convolutional layers to a floating-point of 16 bits or even an unsigned integer of 8 bits could further improve its throughput and will also be studied in future works.

ACKNOWLEDGMENT

The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this paper. URL: <http://sdumont.lncc.br>.

This work was funded in part by FINEP (*Financiadora de Estudos e Projetos* – Funding Authority for Studies and Projects) Projeto FAEV (*Ferramentas Avançadas de Ensaios em Voo* – Advanced Tools for Flight Testing), under Grant Agreement Ref.: 01.22.0115.00.

REFERENCES

- [1] L. E. G. de Vasconcelos, N. P. O. Leite, A. Y. Kusumoto, L. Roberto, and C. M. A. Lopes, "Store separation: Photogrammetric solution for the static ejection test," *International Journal of Aerospace Engineering*, vol. 2019, p. 6708450, Jan 2019.
- [2] D. Hu, D. DeTone, and T. Malisiewicz, "Deep charuco: Dark charuco marker pose estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8428–8436, 2019.
- [3] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and Vision Computing*, vol. 76, 06 2018.
- [4] Itseez, *The OpenCV Reference Manual*, 2.4.9.0 ed., April 2014.
- [5] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, p. 2280–2292, 06 2014.
- [6] Y. Zheng, Y. Kuang, S. Sugimoto, K. Åström, and M. Okutomi, "Revisiting the pnp problem: A fast, general and optimal solution," in *2013 IEEE International Conference on Computer Vision*, pp. 2344–2351, 2013.
- [7] S. D. Blostein and T. S. Huang, "Error analysis in stereo determination of 3-d point positions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 6, pp. 752–765, 1987.
- [8] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming," *Pattern Recognition*, vol. 51, 10 2015.
- [9] O. Araar, I. E. Mokhtari, and M. Bengherabi, "Pdcats: a framework for fast, robust, and occlusion resilient fiducial marker tracking," *Journal of Real-Time Image Processing*, vol. 18, pp. 691–702, Jun 2021.
- [10] G. N. Maria Cristina, V. G. Cruz Sanchez, O. O. Vergara Villegas, M. Nandayapa, H. d. J. Ochoa Dominguez, and J. H. Sossa Azuela, "Study of the effect of combining activation functions in a convolutional neural network," *IEEE Latin America Transactions*, vol. 19, no. 5, pp. 844–852, 2021.
- [11] G. Adriano de Melo, D. N. Sugimoto, P. M. Tasinaffo, A. H. Moreira Santos, A. M. Cunha, and L. A. Vieira Dias, "A new approach to river flow forecasting: Lstm and gru multivariate models," *IEEE Latin America Transactions*, vol. 17, no. 12, pp. 1978–1986, 2019.
- [12] E. Paiva, A. Paim, and N. Ebecken, "Convolutional neural networks and long short-term memory networks for textual classification of information access requests," *IEEE Latin America Transactions*, vol. 19, no. 5, pp. 826–833, 2021.

- [13] R. Gonzales-Martínez, J. Machacua, P. Rotta, and C. Chinguel, "Hyperparameters tuning of faster r-cnn deep learning transfer for persistent object detection in radar images," *IEEE Latin America Transactions*, vol. 20, no. 4, pp. 677–685, 2022.
- [14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [15] S. Guo, Z. Liang, and L. Zhang, "Joint denoising and demosaicking with green channel prior for real-world burst images," *IEEE Transactions on Image Processing*, vol. 30, pp. 6930–6942, 2021.
- [16] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, pp. 261–318, Feb 2020.
- [17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.
- [18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.
- [19] G. Bradski, "The OpenCV Library," *Dr. Dobbs' Journal of Software Tools*, 2000.
- [20] W. FORSTNER, "A fast operator for detection and precise location of distinct points, corners and center of circular features," in *Intercommission Conference on Fast Processing of Photogrammetric Data*, pp. 281–305, 1987.



Gabriel Adriano de Melo Gabriel Adriano de Melo is a Ph.D. student at Instituto Tecnológico de Aeronáutica (ITA) in Electronic and Computer Engineering, where he also received his MSc in 2020 in the same area and BSc in Computer Engineering in 2019. Currently, Melo is a researcher at the Instituto de Pesquisas e Ensaios em Vôo (IPEV) and works with artificial intelligence and information systems applied to flight testing. He worked with NLP during his MSc and currently develops computer vision systems.



control, and artificial intelligence.

Marcos Máximo Marcos R. O. A. Máximo received the BSc degree in Computer Engineering (with Summa cum Laude honours) and the MSc and PhD degrees in Electronic and Computer Engineering from ITA, Brazil, in 2012, 2015 and 2017, respectively. Máximo is currently a Professor at ITA, where he is a member of the Autonomous Computational Systems Lab (LAB-SCA) and leads the robotics competition team ITAndroids. He is especially interested in humanoid robotics. His research interests also include mobile robotics, dynamical systems



Paulo André Paulo André Lima de Castro is an Associate Professor at ITA, where he obtained his B.Sc. in Computer Engineering in 1997. He has received his Ph.D. (2009) degree from Escola Politécnica da Universidade de São Paulo (Poli/USP). He was a post-doctoral visitor at the City University of New York from 2012 to 2013. His main areas of expertise are artificial intelligence, multiagent systems and applications of AI to Finance.