Intelligent vision-based quadrotor path follower

P. Becerra-Martínez[a], D.A. Martínez-Velasco[a], B. Briseño-Tepepa[b] and H. Rodríguez-Cortés[a*]
[a]Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional,
Sección de Mecatrónica, Depto. de Ingeniería Eléctrica,
Ciudad de México, 07360, México
E-mail: {pedro.becerra,dalexis.martinez,hrodriguez}@cinvestav.mx
[b] UPIITA, Instituto Politécnico Nacional
E-mail: bbriseno@ipn.mx

Abstract

Autonomous visual navigation in unmanned aerial vehicles (UAVs) can eliminate the need for global positioning systems and predefined maps, favoring more flexible solutions. This work presents an intelligent vision-based path-following controller for a commercial DJI drone. The controller relies on a Convolutional Neural Network (CNN), developed in Python using TensorFlow-Keras, which is trained to predict the desired drone position from a path marked with tape captured by the camera. The training data is obtained manually from supervised flights with automatic labeling. A nonlinear controller forces the drone to achieve the desired position, allowing it to travel the path. The results obtained using the *Shape Context* descriptor method demonstrate that the proposed strategy achieves effective route tracking.

1 Introduction

Computer vision is now a widely used technology; its application has grown significantly since the 2010s, particularly with the development of the Convolutional Neural Network (CNN) AlexNet, which dramatically reduced the error rate in image classification tasks (ILSVRC 2012) [1]. As a result, computer vision is now routinely applied to a wide range of domains, including object detection [2], facial feature recognition [3], medical image analysis [4], crop monitoring [5], and autonomous navigation [6].

In the context of autonomous systems, computer vision provides a rich, passive sensing modality that enables mobile robots to interpret their environment using visual data. This characteristic is especially valuable for Unmanned Aerial Vehicles (UAVs), where autonomous navigation based on computer vision provides a versatile and cost-effective alternative to traditional localization systems, such as the Global Positioning System (GPS), particularly in urban canyons, indoor settings, or where reliance on satellite-based localization is infeasible. The ability to follow visible paths using only onboard cameras minimizes dependency on external infrastructure.

The backbone for solving autonomous navigation tasks using computer vision is the integration of CNNs that enable robust perception of the environment through learned feature hierarchies, are the driving force behind key capabilities in vision-based navigation such as semantic segmentation [7], monocular depth estimation [8], obstacle detection, and visual odometry [9]. These tasks are crucial for real-time autonomous decision-making, enabling UAVs to map their environment, navigate around obstacles, and plan safe trajectories.

CNNs are used in two main navigation paradigms: Mediated perception approaches, where CNN-based perception modules extract high-level features or semantic maps that inform downstream modules for planning and control [10]. Behavioral reflex approaches, where visual inputs are directly mapped to control commands using deep CNNs trained with imitation learning [11] or reinforcement learning [12].

Path planning is one of the key challenges in autonomous navigation. For example, in [13], the authors address the problem of computing minimum-time trajectories for first-person-view (FPV) drones navigating through cluttered environments. Their approach leverages reinforcement learning to generate time-optimal paths while ensuring obstacle avoidance. In the domain of self-driving vehicles, computer vision techniques utilizing onboard cameras are employed for tasks such as image segmentation and object detection. Specifically, geometric shapes of particular colors are identified through visual processing, and this information is used to estimate the appropriate steering angle, enabling the vehicle to follow a predefined path [14].

Nevertheless, several challenges remain in deploying CNNs for autonomous navigation in the wild. These include generalization across diverse environments, handling sensor noise and occlusion, and real-time inference constraints on embedded platforms. Addressing these issues has led to research in domain adaptation [15], uncertainty-aware modeling [16], and hybrid systems that integrate learning-based perception with traditional robotics pipelines [17].

This work focuses on the design of a vision-based path-

*Authors thank the support of Mexican SECIHTI under project Ciencia de Frontera CF-2023-I-551

following control system for commercial quadrotors employing a CNN as the perceptual core. The system interprets images captured by a front-facing camera to determine the direction to follow, allowing direct visual tracking of a path defined by a colored tape. The CNN is trained under the mediated perception approach to replace the image processing of a navigation strategy based on the center of mass (CMSS) of the viewed path. The implementation is carried out on a DJI mini 3 pro drone, utilizing a laptop as the processing unit, with a solution developed in Python using TensorFlow-Keras. This configuration aims to maximize portability and leverage resources available in the commercial environment.

2 Real-time video stream processing

2.1 Drone video acquisition

The primary tool for communication with the drone is an Android 14 smartphone with the 5.12.0 version of the DJI Mobile SDK (MSDK) [18] pre-installed on it and connected, via a USB Type-C cable, to the DJI remote controller (RC). The MSDK app was modified by adding two virtual buttons for activating and deactivating Virtual Sticks. The Virtual Sticks allow for commanding the drone's translational velocities and rotational speed around the vertical axis.

Two servers are running on an Acer laptop with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 16.0 GB RAM, 3.01 TB storage with an NVIDIA GPU GeForce GTX 1660 Ti: one for collecting video frames using *MediaMTX* [19], an open-source software that allows for publishing video streams with Real-Time Messaging Protocol (RTMP) over a TCP connection on port 1935. The stream is captured in Standard Definition (SD) landscape orientation (848 x 480 px) from the MSDK app, with a maximum frame rate of 30 FPS. The other one is a TCP-gRPC server through which commands to control the drone are transmitted via Wi-Fi from the laptop to the drone radio controller. This experimental setup was designed in [20]. Consult the official drone documentation [21]

2.2 Image processing with OpenCV library

The video frames are captured in the three-channel RGB color format by a down-facing camera integrated into the drone. Since a particular blue tape defines the desired path, the RGB format is converted to HSV format, which separates color tone (H) from intensity (S) and luminosity (V) [22]. Then, color segmentation is implemented by defining a range between $[100, 150, 50]$ and $[140, 255, 255]$ within the HSV space where the blue color of the tape exists. The frame is then cleaned using morphological operations, such as "close" to fill in the gaps formed by black pixels within the white areas, and "open" to eliminate small isolated pixels, since any noise, gaps, or false spots in the mask directly affect the location of that center. Finally, the pixel values are normalized by dividing the mask by 255, resulting in values between 0 and 1. In practice, this process is equivalent to applying a blue color filter, where ones represent the detected path and zeros correspond to the background. The pseudoalgorithm in 1 illustrates the workflow of this image processing procedure, while Figure 1 presents the effect of the image processing on a image frame.

Algorithm 1 Color Detection using CMSS

A Input: frame, lower_color, upper_color, kernel_size
Output: (x_cmss, y_cmss), mask_clean
B hsv ← ConvertColor(frame, BGR_to_HSV)
C color mask ← InRange(hsv, lower_color, upper_color)
kernel ← Ones(kernel_size, kernel_size)
mask closed ← Morph(mask, MORPH_CLOSE, kernel)
mask clean ← Morph(mask_closed, MORPH_OPEN, kernel)
D mask clean ← mask clean
binary mask ← mask clean / 255
if *Sum(binary mask) == 0* **then**
  return None, mask clean
com ← CenterofMass(binary mask)
if *isnan(com.x) or isnan(com.y)* **then**
  return None, mask clean
else
  x_cmss ← int(com.x) y_cmss ← int(com.y)
  return (x_cmss, y_cmss), mask_clean

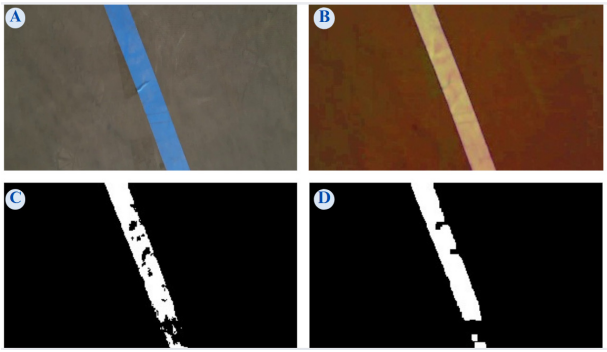

Figure 1: Image processing on a non-augmented sample using Algorithm 1. The input (A), BGR to HSV (B), the color mask (C) and clean mask (D) are shown.

3 Drone model and control

Considering the RC control inputs, the DJI drone flying at a constant altitude can be modeled by the following differential equations [20]. The altitude dynamics are omitted in this model because the drone's onboard control system maintains a nearly constant altitude automatically during flight. This simplification allows the model and controller to focus on horizontal motion, while altitude variations are effectively handled by the drone's internal stabilization.

$$\dot{X} = R_\psi U, \ \dot{R}_\psi = R_\psi S(r) \tag{1}$$

with $X = [\ x \quad y\ ]^\top$ the quadrotor inertial position, $U = [\ u \quad v\ ]^\top$ the quadrotor longitudinal velocity in body

coordinates,

$$R_\psi = \begin{bmatrix} c_\psi & -s_\psi \\ s_\psi & c_\psi \end{bmatrix} \in SO(2)$$

the drone attitude with

$$SO(2) = \left\{ R_\psi \,|\, R_\psi^\top R_\psi = I,\ \det(R_\psi) = 1 \right\}$$

and r the drone angular speed around its vertical axis[1]. Moreover,

$$S(r) = \begin{bmatrix} 0 & -r \\ r & 0 \end{bmatrix} \in \mathfrak{so}(2)$$

with $\mathfrak{so}(2)$ the Lie algebra of $SO(2)$.

At each frame, see Figure 2, it is considered that the inertial reference system is located at the center of the image so that the drone position is $X = 0$. Moreover, the CoM position is defined as the desired drone position X_d and the angle between the $0x^i$ axis and the line that joins the center of the image and the position of the CMSS is equal to ψ.

To ensure compatibility between the image coordinate frame and the drone coordinate frames, all positions are expressed in pixel coordinates. The image coordinate system $0x^v y^v$ is translated to the center by summing to all image measurements $X_c^v = \begin{bmatrix} c_x & c_y \end{bmatrix}^\top$ with $c_x = 424$ and $c_y = 240$, which represent the center of an image of 848×480 pixels.

Using OpenCV the CMSS of the path viewed in the current frame is defined as $X_{CMSS}^v = \begin{bmatrix} cmss_x & cmss_y \end{bmatrix}^\top$, thus

$$X_d^v = X_{CMSS}^v + X_c^v$$

The position error at each frame is defined as

$$\tilde{X} = X_d - X$$

with $X_d = R_{(-\pi/2)} X_d^v$, since at each frame the drone position is assumed to coincide with the image center, it follows that $X = 0$. The angle ψ can be computed as

$$\psi = \arctan\left(\frac{\tilde{y}}{\tilde{x}}\right) \tag{2}$$

The control objective is stated as follows.

Control objective. Assume that X_{CoM} is available. Design U and r such that between image frames $\tilde{X}$ and ψ approach zero. The control objective is satisfied with the following controller. For the translational position

$$U = R_\psi^\top \tilde{X} \tag{3}$$

For the angular position the controller is designed following [23]. The desired attitude is $\psi_d = 0$ so that $R_{\psi_d} = I$. The attitude error is $\tilde{R}_\psi = R_\psi$. Thus,

$$r = -k_r P_a(R_\psi)^\vee \tag{4}$$

with

$$P_a(\tilde{R}_\psi) = \frac{1}{2}\left(\tilde{R}_\psi - \tilde{R}_\psi^\top\right) \in \mathfrak{so}(2)$$

and $(\cdot)^\vee : \mathfrak{so}(2) \to \mathbb{R}$.

[1] In the following, $s_\psi = \sin(\psi)$ and $c_\psi = \cos(\psi)$.

4 Navigation strategies

Once each frame from the video stream is processed, it is used to define a control strategy that guides the drone along the path. The intelligent control strategy was developed as follows. First, the processed image is employed to determine a point over the path that the drone must follow. This point is computed using OpenCV commands such as the CMSS of the detected path with respect to the image center. A yaw angle is also determined as the angle between the line that joins the CMSS location and the center of the image and the vertical axis of a reference frame attached to the image frame. The point location and angle are fed into the controller, which drives the drone to travel along the path. Then, using the data from this navigation strategy, a CNN is trained to determine the point location and angle that are fed to the controller.

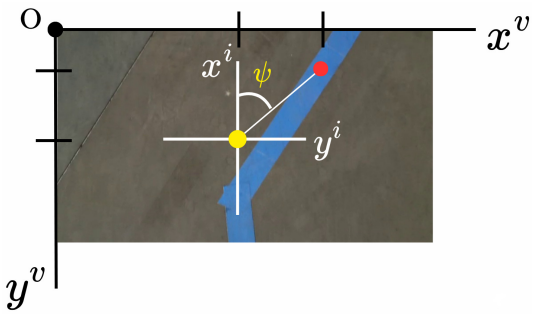

Figure 2: Inertial frame $0x^i y^i$ origin (yellow point). Path CMSS X_d (red point). Yaw angle ψ.

4.1 Navigation without the CNN

The pseudoalgorithm 1 ends with all the information to implement the controller (3), (4). In the results section is shown that the drone is able to track the path.

4.2 Navigation with the CNN

Now, data from four flights of navigation without the CNN were saved; this data consists of an image frame labeled with the corresponding CMSS location and saved to a CSV file organized in three columns. This data was used to train the CNN and compute the path based on CMSS location predictions.

Data collection

From the navigation experiments without CNN, one frame out of every six frames was saved, resulting in a total of 2065 samples organized into four folders: two corresponding to clockwise (CW) route tracking and two to counterclockwise (CCW). This division was necessary because, in practice, the route is not always followed in exactly the same way. By structuring the dataset in this way, we ensured that both directionality and variability in the path execution were adequately represented for training and evaluation.

To strengthen CNN training for color and shape detection in scenarios where luminosity may vary, data augmentation was employed. For each image sample, five

versions of it were generated, two to vary the brightness and one for contrast, using the following relationship [24]:

$$g(i,j) = \alpha \cdot \mathbf{f(i,j)} + \beta$$

where $f(i,j)$ and $g(i,j)$ denote the original and new pixel values at position (i,j), respectively. The parameter $\alpha \in (0,1)$ adjusts contrast, while β controls brightness: $\beta < 0$ darkens the image, and $\beta > 0$ increases brightness.

In the fourth and fifth images the gamma value was modified according to the formula [24]:

$$O = \left(\frac{\mathbf{I}}{255}\right)^{\gamma} \cdot 255 \tag{5}$$

where O are the mapped output values from the image, I are the input values; for $0 < \gamma < 1$ light areas appear darker and $\gamma > 1$ darker areas appear lighter. The CMSS position label was only copied to the extra five variations of each frame.

After data augmentation the dataset has 38,778 image frames and labels. Figure 3 shows the augmented data and the original frame.

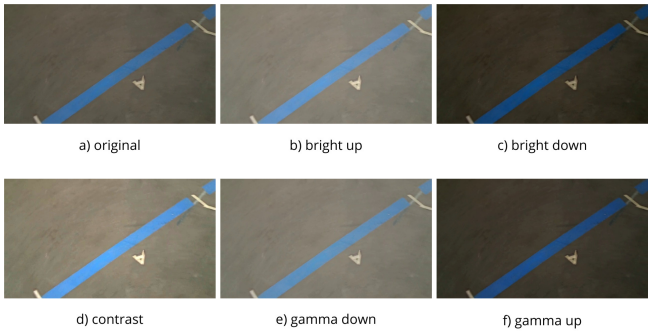

Figure 3: Sample of the dataset after the data augmentation.

4.3 CNN

The CNN was trained with TensorFlow. The input for model is a dataset binarized and resized to 300x169 px, with a single channel (grayscale) normalized in the range $[0,1]$. The structure of the network is composed of three convolutional blocks, see Table 1, with ReLU activation and max-pooling, followed by dense layers that perform the regression of two continuous values, the coordinates of the CMSS X_{CMSS}.

From the augmented dataset, only 20,000 samples were selected for training, due to the size of the processing unit memory in the GPU which is 6 GB. The maximum number of samples was determined by trial and error. The 80% was used for training and the rest for validation. The model was trained for a maximum of 100 epochs with a batch of 16 and the *Adam* optimizer with the default learning rate. Regarding the training errors, the Mean Square Error (MSE) is used as a loss function while the Mean Absolute Error (MAE) is used as an evaluation metric that represents the average deviation between model predictions and actual values. To

	Layer	Details
0	Conv2D (Input)	32 filters, 3×3 kernel, ReLU activation
1	MaxPooling2D	2×2 pool size
2	Conv2D	64 filters, 3×3 kernel, ReLU activation
3	MaxPooling2D	2×2 pool size
4	Conv2D	128 filters, 3×3 kernel, ReLU activation
5	MaxPooling2D	2×2 pool size
6	Flatten	Converts 3D tensor to 1D
7	Dense	128 units, ReLU activation
8	Dense (Output)	2 units (linear activation for regression)

Table 1: CNN Architecture.

avoid overfitting, *Early Stopping* strategies were implemented, and the model was saved with the lowest loss in validation using *Model Checkpoint*. The combination of software and library versions that allowed running the Nvidia GPU for training correctly is presented in Table 2.

Software/Library	Details
Operating system	Windows 11
Python	3.11.4
TensorFlow	2.18.0
Sklearn	1.3.0
NumPy	1.24.3
CUDA Toolkit	12.7
cuDNN	8.6
GPU	NVIDIA GTX 1660 Ti
NVIDIA driver	566.36
VSCode	1.102.1
Anaconda	2.6.3

Table 2: Computational resources for CNN training.

5 Results

5.1 Trained CNN

The training process took 42 epochs to complete, during which the MAE and the loss function were monitored for training and validation sets.

Figure 4 shows the evolution of the MAE. A continuous decrease is observed in the training set, from 0.0330 to 0.0092, indicating that the model undergoes progressive learning from the dataset. The validation set decreases towards an oscillation phase between 0.0169 and 0.0207. This variability suggests that the model reached a saturation point in its capacity to generalize. The model achieved a final MAE of 0.0154 over normalized coordinates, equivalent of $\approx$ 13 px in X and $\approx$ 7 px in Y, corresponding to an average Euclidean distance of $\approx$ 15 px.

Figure 5 corresponds to the loss function (MSE) during the training process. From epoch 10 onwards, training loss decreases until it stabilizes around 0.0006. In the validation set, it stabilizes without showing relevant changes, which suggests that the model does not incur significant overfitting and maintains its generalization capacity throughout training. The final loss (MSE) was 0.0012, whose root mean (RMSE) corresponds to an er-

ror of ≈ 29 px in X and ≈ 17 px in Y, with an average Euclidean distance of ≈ 34 px on the validation set. The model learned around epoch 15–20, suggesting that continuing training beyond this point offers little improvement. Model performance was confirmed by real-world testing.

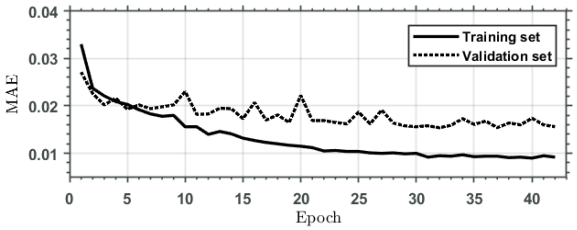

Figure 4: MAE vs Epoch graph.

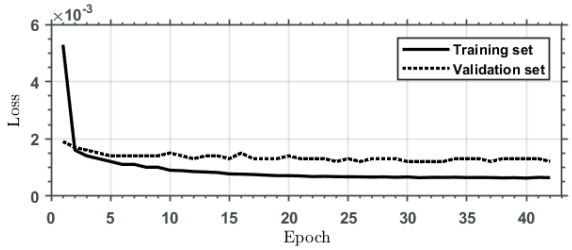

Figure 5: Loss (MSE) vs Epoch graph.

5.2 *Experimental tests*

Two runs were performed using the CMSS calculation and two more using the predicted values from the model, CW and CCW. Figure 6 and Figure 7 were obtained from these runs, where the trajectories were plotted using data from an *Optitrack* system.

To verify the following of the real path, the *Shape Context* algorithm was implemented, which is scale invariant and reflects the structural difference between shapes [25], estimating correspondences between points in the geometry paths traced by both, predictions from the CNN and CMSS. The result indicates that with the CNN prediction method with a shape context distance of 64.6561 (CCW) and 123.3177 (CW) it has better similarity with the original trajectory, while the shape context distance of the CMSS was 71.5434 (CCW) and 133.6464 (CW) making it further away in geometric structure.

6 Conclusions

This work presented the design of a control algorithm calculated from CNN predictions of coordinates so that a commercial DJI drone can navigate autonomously on a reference, using freely available software tools that can be used for real-time image processing with computer vision. The system was validated through experiments under real-world conditions. In the initial stage, tracking issues were identified due to light reflections in areas on the floor where the blue tape was placed. This led to increasing the training dataset by varying the illumination range of the images already taken. The capabilities of the proposed system were tested and met the objective, as can be seen in the comparative graphs in Figure 6 and Figure 7 of the experimental routes and the difference calculated with the Shape context algorithm. The result is that the predictions of a trained model combined with computer vision can be an alternative for solving real-time autonomous navigation tasks. To complete the experiment, a full analysis of the CNN used is currently underway. Consideration is being given to strengthening the system by using deeper networks such as CNN+LSTM, thus capturing the temporal evolution of the tracking to provide early predictions. This would open up the possibility of integrating more advanced predictive controllers.

Suplementary material at:`https://youtube.com/playlist?list=PLIFqXxbt6YQT_ZDaQDj3KNkFF3OEGUaGR&si=-vMcyr-04_M1mLYf`

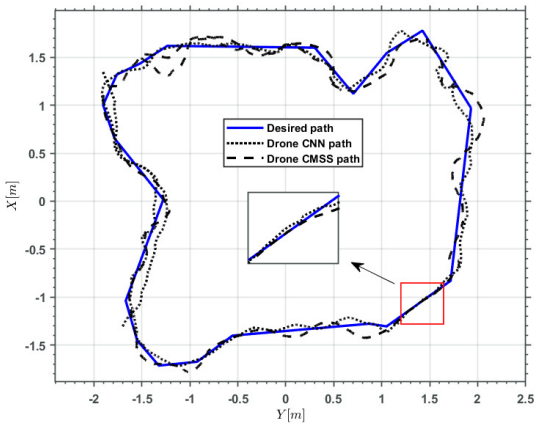

Figure 6: Path traveled CW.

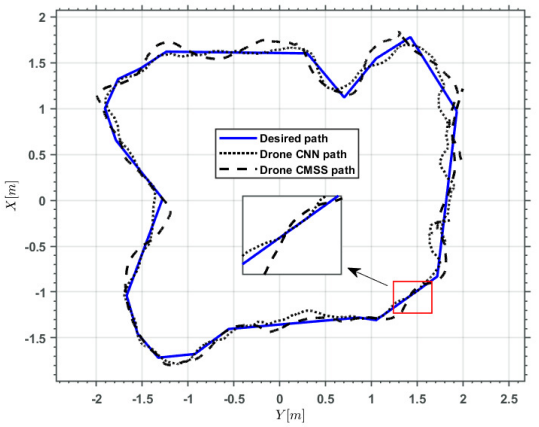

Figure 7: Path traveled CCW.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] Y. Xiao, A. Jiang, J. Ye, and M.-W. Wang, "Making of night vision: Object detection under low-illumination," *IEEE Access*, vol. 8, pp. 123075–123086, 2020.

[3] K.-C. Liu, C.-C. Hsu, W.-Y. Wang, and H.-H. Chiang, "Real-time facial expression recognition based on cnn," in *2019 international conference on system science and engineering (ICSSE)*, pp. 120–123, IEEE, 2019.

[4] J. Olveres, G. González, F. Torres, J. C. Moreno-Tagle, E. Carbajal-Degante, A. Valencia-Rodríguez, N. Méndez-Sánchez, and B. Escalante-Ramírez, "What is new in computer vision and artificial intelligence in medical image analysis applications," *Quantitative imaging in medicine and surgery*, vol. 11, no. 8, p. 3830, 2021.

[5] D. Story and M. Kacira, "Design and implementation of a computer vision-guided greenhouse crop diagnostics system," *Machine vision and applications*, vol. 26, no. 4, pp. 495–506, 2015.

[6] D. Lee, G. Kim, D. Kim, H. Myung, and H.-T. Choi, "Vision-based object detection and tracking for autonomous navigation of underwater robots," *Ocean Engineering*, vol. 48, pp. 59–68, 2012.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[8] C. Godard, O. M. Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6602–6611, 2017.

[9] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, p. 2043–2050, IEEE, May 2017.

[10] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2722–2730, 2015.

[11] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[12] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8248–8254, 2019.

[13] R. Penicka, Y. Song, E. Kaufmann, and D. Scaramuzza, "Learning minimum-time flight in cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7209–7216, 2022.

[14] S. Shafique, S. Abid, F. Riaz, and Z. Ejaz, "Computer vision based autonomous navigation in controlled environment," in *2021 International Conference on Robotics and Automation in Industry (ICRAI)*, pp. 1–6, IEEE, 2021.

[15] Y. Zhang, P. David, H. Foroosh, and B. Gong, "A curriculum domain adaptation approach to the semantic segmentation of urban scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 1823–1841, 2020.

[16] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," *Advances in neural information processing systems*, vol. 30, 2017.

[17] F. Atas, G. Cielniak, and L. Grimstad, "Elevation state-space: Surfel-based navigation in uneven environments for mobile robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5715–5721, 2022.

[18] DJI Developers, "Dji mobile sdk documentation - introduction." `https://developer.dji.com/mobile-sdk/documentation/introduction/index.html`, 2025.

[19] GitHub Repository:bluenviron/mediamtx, "mediamtx: Ready-to-use rtsp server and media proxy." `https://github.com/bluenviron/mediamtx`, 2025.

[20] M. Martínez-Ramírez, M. Trujillo-Flores, X. Shao, J. G. Romero, and H. Rodríguez-Cortés, "A collision avoidance strategy for commercial quadrotors," in *2025 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 988–993, 2025.

[21] DJI, "Mini 3 pro - product support." `https://www.dji.com/mx/support/product/mini-3-pro`, 2025.

[22] OpenCV Documentation, "Changing colorspaces — opencv documentation." `https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html`, 2025.

[23] H. Rodríguez-Cortés and M. Velasco-Villa, "A new geometric trajectory tracking controller for the unicycle mobile robot," *Systems & Control Letters*, vol. 168, p. 105360, 2022.

[24] OpenCV Documentation, "Basic linear transforms — opencv documentation." `https://docs.opencv.org/4.x/d3/dc1/tutorial_basic_linear_transform.html`, 2025.

[25] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 4, pp. 509–522, 2002.