Leveraging a Neural Pilot via Automatic Gain Tuning using Gate Detection for Autonomous Drone Racing

L. Oyuki Rojas-Perez, J. Martinez-Carranza*
Instituto Nacional de Astrofísica, Óptica y Electrónica Puebla, Mexico

Abstract

Autonomous Drone Racing (ADR) has pushed the development of artificial pilots that enable a drone to fly autonomously on a race track. DeepPilot is an artificial pilot trained with a Deep Learning (DL) methodology for ADR. Given a set of camera images, this neural pilot estimates flight signals to navigate the drone throughout the race track. However, DeepPilot requires noise filters in output values to smooth out peaks in the flight signals, which produce oscillatory behaviour and jolts. Seeking to improve the performance of the artificial pilot, we propose to use a gate detector, also based on DL, to detect gates and use the position of the gate on the image in a state machine that automatically calculates the corresponding gains. To assess our approach, we carried out experiments in a simulation environment with a racetrack composed of 18 gates at different heights and orientations. We present a comparison with manually-set gains versus our automatic gain tuning approach. With the latter, the drone completes more than 15 consecutive laps on the same racetrack without collisions. Our approach was also successful in the scenario where some gates change position dynamically.

1 Introduction

Recent solutions to Autonomous Drone Racing (ADR) have proposed to use a deliberative approach involving path planning, control and trajectory tracking via Model Predictive Control and Reinforcement Learning to find optimal trajectories and control flight signals to navigate a race track as fast as possible [1, 2, 3]. Some of these approaches have been successful in demonstrating performance comparable to a human pilot [4]. However, this strategy relies on high-frequency and accurate pose estimation of the drone and knowledge of the position of the gates in the race track.

In contrast, other approaches have explored a more reactive strategy where no drone position or race track knowledge is available to the artificial pilot. Instead, the latter is *trained* to regress instantaneous flight signals from an image captured with a camera on board the drone [5, 6]. Deep Learning has become the viable option for implementing neural models that learn flight signals from camera images for ADR. For example, in our previous work, DeepPilot [6], we present an artificial pilot based on a Convolutional Neural Network (CNN). That includes temporal information for flight signal estimation by using six images equally sampled in one second (i.e., every five images per second). These images form a single image mosaic associated with their corresponding flight signals.

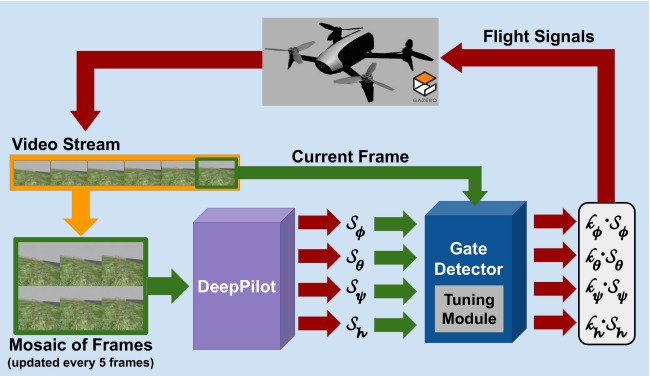


Figure 1: Schematic view of our approach where we propose to use a gate detector to leverage the performance of DeepPilot [6], a neural pilot trained to estimate flight signals $(S_\phi, S_\theta, S_\psi, S_h)$ from camera images. We propose to use the gate position detected on the current image to automatically tune a set of gains $(K_\phi, K_\theta, K_\psi, K_h)$ to replace the noise filter used by DeepPilot to avoid oscillatory behaviour and signal saturation. A video illustrating our approach can be seen at `https://youtu.be/49OzJCHTJu4`

DeepPilot has been shown to enable a drone to navigate on a race track with gates randomly placed and at different orientations and heights. However, the estimated flight signals should be noise filters in output values to smooth out peaks in the flight signals, which produce oscillatory behaviour and jolts. Another disadvantage is that if we change the platform, for example, the Bebop 2 simulated in RotorS [7], DeepPilot provides the incorrect proportion of the signal value. It means that the estimated flight signals are correct regarding what flight signal has to be sent to the drone (angle in roll, angle in pitch, velocity in yaw, velocity in altitude)

*Department of Computer Science at INAOE. Email addresses: {oyukirojas, carranza}@inaoep.mx

but with an incorrect proportion of the signal value. There are two options to solve this: 1) tuning a set of constants that act as *gains* to improve the performance of the artificial pilot and 2) providing more examples to the training set. However, this would involve the generation of large data sets and long training times.

Motivated by the fact that gain tuning would be preferable over large data generation, we propose using a gate detector's output to tune these gains automatically. DL has also been effectively used to solve the problem of gate detection on an image in the context of ADR [8], including its use to implement a flight controller [9]. However, given that the gate detector does not provide 3D information about the gate, we can not use the controller to control the yaw signal when the gate is observed in a skew view. Nevertheless, DeepPilot has learned to estimate a yaw signal under this scenario. Thus, we propose to use the gate detector, implemented with an SSD-based model [10] to leverage the performance of DeepPilot via automatic gain tuning. Figure 1 illustrates our full approach: first, flight signals are obtained with DeepPilot, which receives as input a six-image mosaic. Next, the gain tuner receives DeepPilot's output and the output of the gate detector, which uses the current image to detect the gate, whose position will calculate the value of the corresponding gain for each flight signal. Finally, we will send the product of the estimated flight signals multiplied by the gains to the drone.

We performed several experiments in simulation comparing the performance of DeepPilot without gain multiplication, DeepPilot with gains set manually, and DeepPilot with gains tuned using our proposed approach. Our results indicate a significant improvement over the former options. We also include an experiment where the drone performs several flights on the same track, and in some laps, the gates are changed in position to assess whether the gains adjust adequately, which effectively happened as expected.

For a more detailed explanation of our approach, this paper has been organised as follows: Section 2 summarises relevant related works; Section 3 presents our proposed methodology; our experimental framework is described in Section 4; conclusions and future work are discussed in Section 5.

2 RELATED WORK

Most ADR techniques are based on sensing, perception tasks, control, path flight, and localisation relative to the gate or global localisation using external sensors [1]. In most strategies, cameras were the principal sensor used to interpret the environment, gate or obstacle detection and localisation of the vehicle on the race track.

However, the latency from one frame to the next affects the vehicle's reaction to the race track and its speed. This latency and the blind spots (when the drone crosses the gate) are why the authors use complementary information obtained from IMU [9, 11, 12, 13, 14], LiDAR sensors [15, 8, 1], ultrasonic sensors and optical flow [1].

Concerning perception tasks, a popular open-source computer vision library, OpenCV, has been used to perform colour and contour detection. However, these techniques are susceptible to lighting conditions. Some authors [15, 8, 1, 10, 16, 17] have replaced computer vision techniques with deep learning, using open source libraries such as Caffe, Tensorflow and Keras.

Another important task for ADR is the flight path calculation. For this is necessary to use localisation systems such as SLAM, Visual Odometry, Semi-direct Visual Inertial Odometry (SVIO) or position estimation methods based on the IMU [1], using either the Kalman Filter or the Extended Kalman Filter. Even the trajectory planning can be modified by the gate location [15, 8, 1, 10], relative position [18, 13, 19, 9, 20], actions [1], velocity, or even the drone's direction [18, 16].

In contrast to visual localisation, authors in [4] present a solution using external sensors. In this, a drone controlled by artificial intelligence flew faster than three human pilots on a race track, reaching a maximum speed of 16 m/s. However, this approach requires the gates' global position to generate a time-optimal trajectory through the race track and execute the flight path inside a specific area where 36 VICON cameras are set. Therefore the flight path must be pre-calculated and is only useful for static race tracks.

The authors in [21] highlight that the most significant difference between human pilots and artificial pilots guided by a model for trajectory planning and control is that human pilots continuously explore and learn from experience and improve the visibility of gates by choosing an entry angle. In contrast, the based trajectory planning and control approaches do not consider experiences. Another important difference is that the artificial pilot relies on a position tracking system while human pilots perform a vision-based state estimation.

According to the works discussed in this section, we conclude that the solutions presented at ADR have difficulties interacting with dynamic environments and do not consider experience. Instead, most of the works assume that the track is static or use external sensors that allow them to optimise a set of waypoints corresponding to the global gate position. Motivated by this; we propose an approach based on visual information to leverage the behaviour of a neural pilot called DeepPilot and a Single Shot Detector for gate detection. For this approach, we don't require a model for trajectory planning, a global position nor a control methods to guide a drone on a racetrack.

3 METHODOLOGY

3.1 Flight Signal Estimation

DeepPilot is a computational model that associates a set of images with a flight signal. These signals represent the angular position of the drone body structure at the roll and pitch angles, thus producing a translational motion at those angles, the rotational speed at the yaw angle, and the verti-

cal speed referenced to altitude. The values of these 4 flight signals $(S_\phi, S_\theta, S_\psi, S_h)$, estimated by the DeepPilot model, are passed to the drone's internal controller, thus allowing the drone to navigate autonomously through the race track gates, assuming that the next gate becomes visible immediately after crossing the current gate.

The DeepPilot architecture comprises three parallel branches to infer roll and pitch together but yaw and altitude separately. Each branch has four convolutional layers with three inception modules, one fully connected layer, and a regressor layer.

3.2 Gate Detector

We use the Single Shot Detector (SSD) network [22], a fast multi-category object detector, for gate detection. The SSD combines predictions on multiple feature maps of different sizes, producing detections at high and low levels of the image by applying convolution filters. The SSD network architecture is based on a VGG16 image classification network and an auxiliary structure composed of multiple convolution layers. The auxiliary structure obtains multi-scale feature maps and convolutional predictions to know the bounding box displacement and the box's position relative to the location of each feature map.

In this work, we use a reduced variant called SSD7; this is a small optimised network that uses an auxiliary structure composed of 7 convolution layers, which reduces the training and search time. The model used to detect the gate in a racetrack was trained with images obtained from the Gazebo simulator, and real environments, indoors and outdoors [9, 10].

3.3 Gain Tuning

Figure 1 shows the general diagram of our approach. First, the gazebo video stream from the Bebop2 platform camera is acquired to generate a mosaic composed of six frames updated every five frames, used as input to DeePliot. This mosaic allows DeepPilot to obtain temporal information from the environment to estimate the four flight signals $(S_\phi, S_\theta, S_\psi, S_h)$. It is important to mention that for this work, we will use the raw DeepPilot signals; we do not implement the noise filter presented in [6]. Next, the Gate Detector requested the four flight signals and the current frame of the video stream. In this block, the SSD7 network detects the gate on the image. Then based on the detections, the rules set in the gain tuning module are executed. If the Gate Detector does not provide detection, the following base gains $k_\phi = 1$, $k_\theta = 0.5$, $k_\psi = 5$ and $k_h = 1.2$ are used, obtained by empirical experimentation. Finally, the gains are multiplied by the corresponding flight signals estimated by DeepPilot to send to the Bebop2.

Our aim is to perform automatic gain tuning using the SSD7 gate detector. To this end, we identify four cases during the flight on the racetrack: 1) slow translation at the front, 2) errors on the edge of the image that corresponds to roll values, 3) roll oscillations at the front of the gate and 4) slow

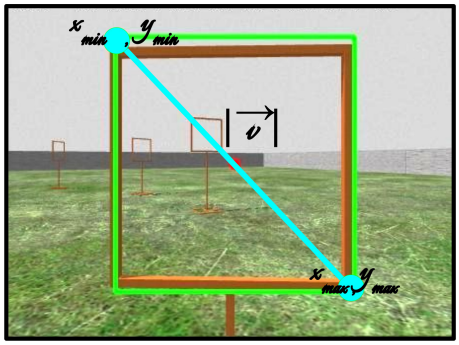


Figure 2: Information provided from the gate detector, where $x_{min}, y_{min}, x_{max}$ and y_{max} are used to select the closest next gate by calculating the largest vector norm $|\overrightarrow{v}|$.

rotation when the gates are skewed. To solve deal with each one of these cases, we design rules for k_ϕ, k_θ and k_ψ based on the gate detection. First, we select the next closest gate on a racetrack and for this, we calculate the vector norm $|\overrightarrow{v}|$ of the detections as shown in Equation 1, where $x = [x_{min} - y_{min}]$ and $y = [x_{max} - y_{max}]$ are obtained from the gate bounding box, then we select the largest vector norm, as shows Figure 2.

$$|\overrightarrow{v}| = \sqrt{x^2 + y^2} \tag{1}$$

We define the first case by using the gate detection area. For example, under the following rule, if the centroid of the gate detection is on the limit indicated in Figure 3, we obtain the value of k_θ using Equation 2, where Det_{area} is the area of the current detection and the $area_{max}$ has a value of 70 (set manually). With this condition, k_θ has a higher value when Det_{area} is smaller. Therefore the vehicle flies faster toward the gate and decreases k_θ value when the area increases. Another condition that affects the k_θ value is when the centroid of the gate detection is on the region indicated by Figure 3 and when S_ϕ and S_ψ provided by DeepPilot are equal to zero; in this case, k_θ has a value of two (twice the speed).

$$k_\theta = \frac{area_{max}}{Det_{area}} \tag{2}$$

The second case is that where the DeepPilot model does not correctly provide the roll signal when the gate is too close to the horizontal image edge, see Figure 4. Even when the gate is inside the viewing area, DeepPilot indicates $S_\phi = 0$. For this reason, we designed the following rule, when the centroid of the gate is at the edges of the image and the S_ϕ is equal to zero, the S_ϕ signal is replaced by 1 or -1 depending on the which edge the gate was spotted. In this way, the drone is forced to respond in the direction of the gate. With this signal, the gate is again inside the viewing area, the zone where DeepPilot provides S_ϕ with a non-zero value.

The third case is when oscillations of S_ϕ provoke the following issues: the vehicle stays in front of the gate for a long

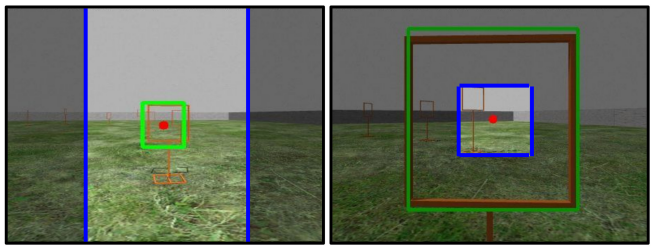


Figure 3: Example of cases to obtain the value of k_θ. (a) If the detection is within the boundary indicated by blue lines, the value of k_θ is obtained using Equation 2. (b) if the centroid of the detection is within the blue box, the gains are equal to $k_\theta = 2$, $k_{\phi=0}$ and $k_{\psi=0}$, which seeks to accelerate the forward motion twice the value estimated by DeepPilot for S_θ

.

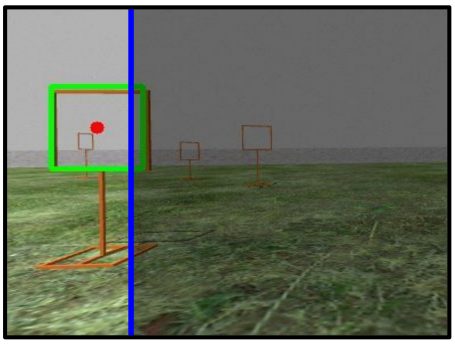


Figure 4: Example of when the gate is near the edge of the image, DeepPilot assigned $S_\phi = 0$ even though the gate is inside the viewing area. For this reason, we replaced the flight signal S_ϕ by 1 or -1, depending on what edge the gate is located, to force the drone to respond in the direction of detection.

time moving from one side to the other, ultimately colliding with the gate edge, or flying outside the gate. We mitigate this by defining k_ϕ with Equation 3, where d is the distance in pixels between the centroid of the detection and the centre of the image, as we showed in Figure 4. Note that d is normalised to half the image's width. Thus the value of k_ϕ is proportional to d, which helps reduce the roll signal provided by DeepPilot.

$$k_\phi = \frac{d}{320} \tag{3}$$

The last case is the slow rotation when the gates are skewed, as shown in Figure 6. We obtain the value of k_ψ using Equation 4 to adjust the rotation signal depending on the width of the bounding box detecting the gate. Here, w is the width of gate detection, which is subtracted from 1 and normalised by the height to obtain an inverse ratio to w.

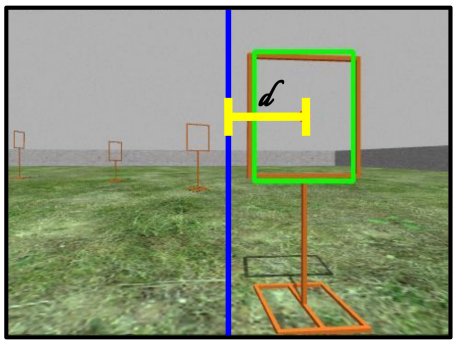


Figure 5: Case 3 used to automatically tune the gain for the roll signal. To this purpose, the distance in pixels from the centroid of the detection and the centre of the image is calculated. Half the image width normalises this distance, giving a ratio that reduces the value of the roll signal provided by DeepPilot. This variable gain helps to reduce oscillations when the drone attempts to centre w.r.t. to the gate.

$$k_\psi = \frac{1.0 - w}{640} \tag{4}$$

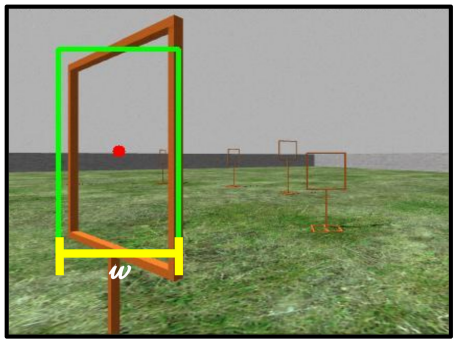


Figure 6: Automatic tuning for the yaw signal under skew views of the gate. This case occurs in turns of the track, mainly due to the proximity between one gate and another. Due to the drone's inertia, the camera has an inclined view of the gate, thus producing a skew view of the gate. Even when DeepPilot estimates a yaw signal, its value may not be enough to align the drone on time. This is overcome with the automatic gain tuning for this case when setting the gain as an inverse function of the width of the bounding box of the detected gate, as shown in Equation 4.

4 EXPERIMENTS

4.1 System Overview

Our proposal was implemented on the Alienware R5 laptop, which has a corei7 processor, 32GB of RAM and an NVIDIA GTX 1070 graphics card. It runs the Ubuntu 18.04 LTs operating system and ROS Melodic Morenia. We used

the Gazebo 9 simulator and the rotors_simulator package to emulate the Parrot Bebop 2 vehicle for the experimental framework; we should highlight that in RotorS[7], the Bebop 2 model exhibits a more agile flight (closer to reality) than that used in the TUM simulator [23]. For the execution of the SSD7 and DeepPilot networks, we used the TensorFlow 1.15 and Keras 2.3.1 frameworks.

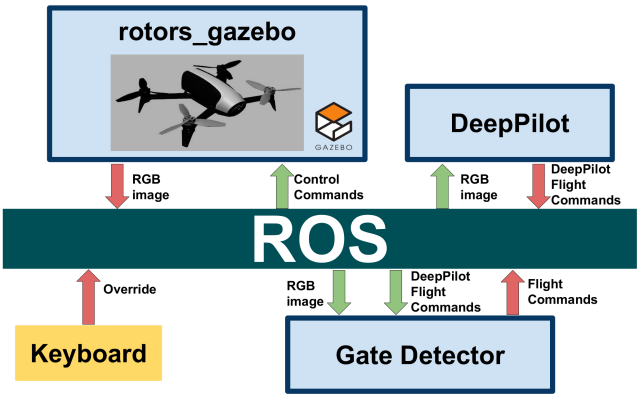


Figure 7: Communication system based on the Robotic Operating System. Green arrows indicate data published by the nodes. Red arrows indicate data consumed by other nodes.

Figure 7 shows the communication architecture used in this work. It shows the interaction between the ROS nodes. Where the principal node is rotors_simulators, which provides information from the camera onboard the Bebop2 at 80 fps, the odometry and simultaneously receives information to control the Bebop2. The DeepPilot node gets the image from Bebop2, generates an image mosaic updated every five frames and provides four flight signals $(S_\phi, S_\theta, S_\psi, S_h)$ at 25 fps. The Gate Detector node receives the Bebop2 image, and we applied the trained model for gate detection. It also gets the flight signals provided by DeepPilot and publishes the flight signals with the automatic gain tuning at 80 fps. Finally, the node Keyboard is used only to initiate or cancel the autonomous flight.

4.2 DeepPilot vs DeepPilot with fixed gain

We use the evaluation racetrack in [6], which is composed of 18 gates. The racetrack extends over $62m \times 44m$, and the spacing between gates is 7m to 9m. Therefore, gates 1, 2, 3, 5, 9, 12 and 14 are 2m high, gates 4, 7, 8, 11, 13, 15, 16 and 18 are 2.5m high, and gates 6, 10 and 17 are 3m high, all of them are placed in an ellipse shape to vary the orientations.

We performed experiments on the Bebop2 platform of the rotors_gazebo simulator using the model trained on the same track using the Ar drone platform of the tum_simulator simulator[1], see Figure 8, where DeepPilot did not perform well as it failed to cross the 18 gates of the race track. DeepPilot

[1]https://github.com/QuetzalCpp/DeepPilot/tree/Noetic

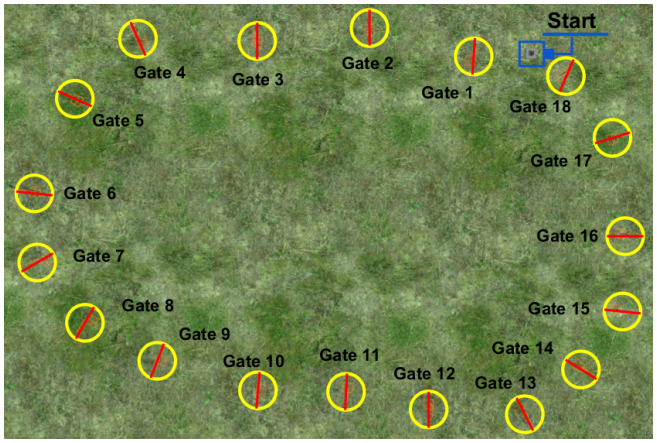
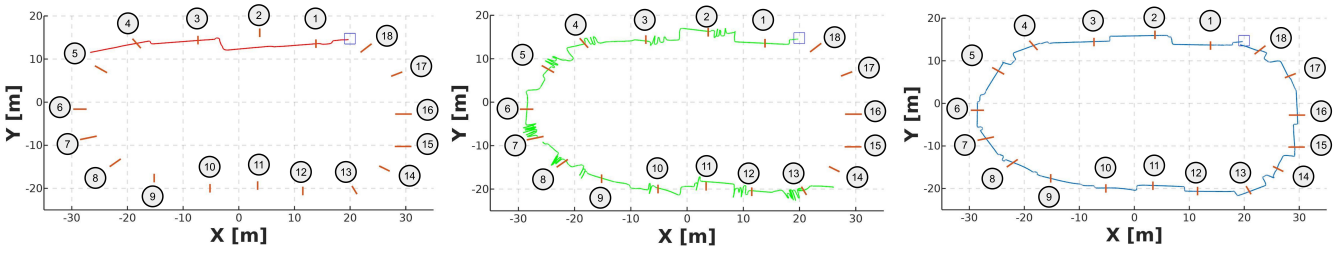


Figure 8: The track consists of 18 gates set at different heights and orientations. The racetrack extends over 62m and has gate spacing of 7m to 9m. Image is taken from [6].

only managed to cross 3 out of 18 gates in ten runs. In this case, the drone always evaded gate number two, as shown in Figure 9(a). Therefore, we analysed the flight signal values obtained during the ten trajectories, where the average is 0.023 for roll, 0.306 for pitch, 0.008 for yaw and 0.006 for altitude. These values indicate that the pitch is too high compared to the rest of the flight signals, which does not allow the drone to cross all the gates in the track.

Therefore, via several flight experiments, we found a set of fixed gains for which, after multiplying by the flight signals estimated by DeepPilot, produced a reasonable performance. This is, the drone managed to cross almost all the gates, although it always hit or flew outside some gates. Thus, with this empirical analysis, we set the gains to be $k_\theta = 0.5$, $k_\phi = 1$, $k_\psi = 5$ and $k_h = 1.2$.

The fixed gains enabled the drone to cross 12 of 18 gates, with gate number 11 being evaded. This can be appreciated in Figure 9(b), where we can see that the drone positioned itself in front of the gate throughout the trajectory. However, the variation of values generated by the roll signal causes an oscillation that stops the drone from crossing the gate. This can be seen in 2 cases, the first at gate 11, where it evades to the left and does not cross the gate, and the second at gate 14, since due to the oscillations the drone cannot correctly generate a signal to yaw. We performed 10 runs using these fixed gains to obtain the following average flight signal values: 0.020 in roll, 0.095 in pitch, 0.012 in yaw and 0.004 in altitude. These values indicate that small values of the pitch signal improves DeepPilot's performance in guiding the drone on the racetrack. However, the oscillations of the roll signal do not allow the drone to complete the track.



(a) (b) (c)

Figure 9: Top view of 1 out of 10 runs in our experiments: (a) corresponds to the drone's trajectory using directly the flight signals provided by DeepPilot; (b) corresponds to the drone's trajectory using the flight signal provided by DeepPilot multiplied by fixed gains set up manually; (c) corresponds to the drone's trajectory using the flight signals provided by DeepPilot multiplied with gains tuned automatically by using gate detection, note that in this case the drone is able to traverse all the gates and finish the lap. A video illustrating these experiments can be seen at `https://youtu.be/49OzJCHTJu4`

(a) (b) (c)

Figure 10: Examples of DeepPilot failing when the gate is at the edge of the image. The top images are snapshots of the 6-image mosaic that DeepPilot uses to estimate flight signals. The inferior pictures are snapshots of the drone and the gate in the gazebo world.

4.3 *DeepPilot with automatic gain tuning using gate detection*

Due to the instability of the drone to complete the race-track, we employed the gate detection for automatic gain tuning under the rules set out in Section 3.3. DeepPilot operates using the $k_\theta = 0.5$, $k_\phi = 1$, $k_\psi = 5$ and $k_h = 1.2$ as default values but these are modified when the SSD7 network detects the nearest gate. Only then do the gains change their value to modify the drone's behaviour guided by DeepPilot.

In Figure 9(c), we note that case 1 is activated on gates 3-4, 5-6, 8-9, 9-10 and 16-17; for instance, the value of k_θ increases, as the gates are far away from each other. Case 2 is activated at the exit of gates 4, 13, 14 and 18, as the neighbouring gates are on a curve. By changing the roll value, the drone is forced to continue on its way. This failure by DeepPilot is attributed to the lack of examples for this type of view. The effectiveness of case 3 is noticeable along the track since the trajectory does not show large oscillations compared to the performance of DeepPilot with fixed gains, see Figure 9(b). Finally, case 4 is activated at gates 8, 14 and 18, as the previous gates are close, exhibiting skew views. In this case, the rule using the the gate detection helps to increase the yaw signal in order to align the drone parallel to the gate.

In Table 1, we compare the performance of DeepPilot against DeepPilot with fixed gains and DeepPilot with automatic gain tuning using gate detection. Note that DeepPilot, with no gains, failed to complete the track, and this is because, on average, the pitch value is higher than the other

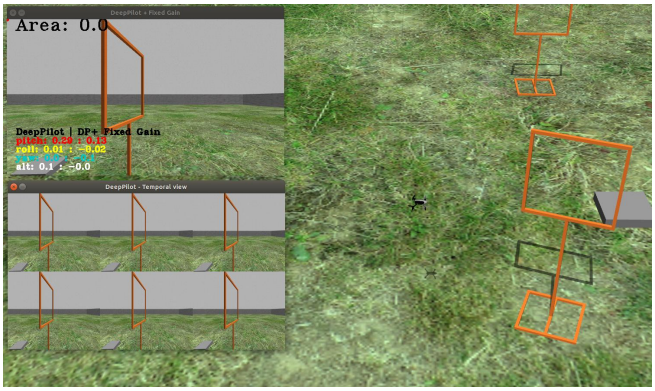


Figure 11: Examples of DeepPilot failing when the gate exhibits a skew view.

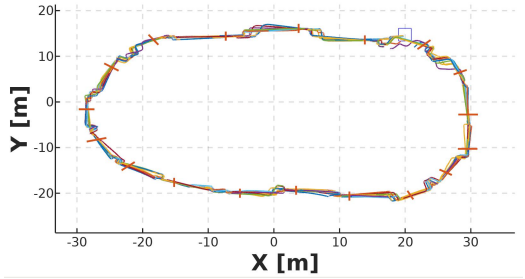
flight signals, making it impossible for the drone slow down and allow the necessary time for the other signals to get estimated properly. When using fixed gains, DeepPilot can cross up to 12 gates.

Nevertheless, the roll oscillations make it fail to cross all the gates. In addition, DeepPilot fails in cases where the gate is at the edge of the image, see Figure 10 or with skew views, as shows Figure 11. Thus, these experiments show the benefits of the automatic gain tuning, which enable the drone to finish the lap. Note that the average pitch is high, meaning that the automatic tuning is effective to assign a high gain value in those cases when the gate is far away or close and centred on the image.

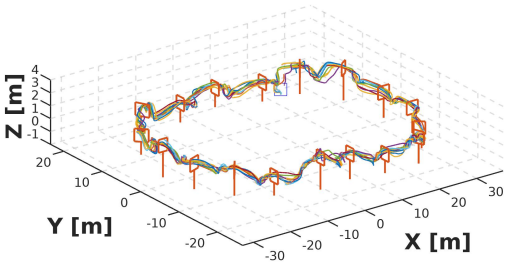
For a time comparison and given that DeepPilot with fixed gains failed after crossing gate 13, the last column in Table 1 shows the average flight time up to gate 13 for DeepPilot with fixed gains and DeepPilot with automatic tuning. Note that the former took a longer time to reach gate 13. This indicates that the automatic gain tuning also helped to reduced the flight time.

A video illustrating these experiments can be seen at `https://youtu.be/49OzJCHTJu4`

In Figure 12(a), we show the performance of DeepPilot with automatic gain tuning using gate detection in 10 runs. We can see that the drone behaves in a stable and repetitive manner. Minimal oscillations in altitude are observed, see Figure 12(b), where the average of the flight signals values is: 0.007 in roll, 0.211 in pitch, 0.038 in yaw and 0.007 in altitude. Finally, Figure 13 shows that our proposal generates a stable and consistent flight that kept the drone flying during 3 hours, performing 18 alps without colliding with the gates, thus obtaining an average time of 9.81 minutes per lap. In these consecutive flights, the average value of the flight signal is 0.004 in roll, 0.217 in pitch, 0.041 in yaw and 0.001 in altitude. This indicates that DeepPilot + GateDet got its flight signals adjusted, avoiding abrupt changes during each lap. Note that this behaviour resembles that of a human pi-



(a)



(b)

Figure 12: Top and side view of the trajectories for 10 runs using the flight signals provided by DeepPilot with automatic gain tuning using gate detection. Note that the drone behaves stably and repetitively. Minimal oscillations in altitude are observed.

lot who improves his performance after each lap. This is an interesting phenomenon that we will analyse in further study.

Finally, Figure 14 shows two runs using DeepPilot automatic gain tuning in two scenarios. In the first one, shown in light blue, it performs the flight with fixed position of the gates in the race track. For the second scenario, indicated in orange, gates 1, 2, 3 change position three times, 5, 8, 10 twice, 11, 12, 16 and 17 change only once. These variations are indicated with dark blue lines. This experiments demonstrates that our proposal works effectively under dynamic changes in the race track.

5 Conclusion

We have presented an automatic gain tuning approach for a neural pilot called DeepPilot developed for Autonomous Drone Racing. DeepPilot estimates flight signals to enable the drone to cross the gates in the race track. However, the estimated flight signals should be noise filters in output values to smooth out peaks in the flight signals, which produce oscillatory behaviour and jolts. Therefore, in this work, we proposed using a gate detector to tune such gains automatically. Usually, a gate detector in ADR has been used to implement a flight controller. However, the detector does not provide 3D information about the gate; thus, such a controller could not be used to control the yaw signal when a gate is observed in a skew view. In contrast, DeepPilot has learned to estimate a

Table 1: Comparison of the performance of DeepPilot against DeepPilot with fixed gains and DeepPilot with adjustable gains using gate detection in a race track of 18 gates. The values $(S_\phi, S_\theta, S_\psi, S_h)$ correspond to the average of flight signals sent to the drone over ten trajectories. Also, we report the number of gates crossed in ten runs and the average flight time up to gate 13 for comparison. Note that the automatic gain tuning helps to reduce the flight time in a half.

Method	Gains	Avg. Flight Signal				Avg. Crossed Gates	Avg. Time up to Gate 13 [min.]
		S_ϕ	S_θ	S_ψ	S_h		
DeepPilot	-	0.023	0.306	0.008	0.006	3/18	-
DeepPilot	Fixed	0.020	0.095	0.012	0.004	12/18	15.20
DeepPilot + Gate Det	Automatic Tuning	0.007	0.211	0.038	0.007	**18**/18	**7.15**

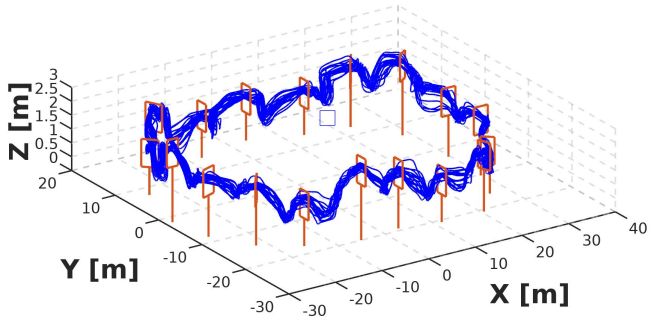


Figure 13: Trajectories of 18 consecutive flights without landing between laps. The drone uses the flight signals provided by DeepPilot multiplied with gains tuned automatically by using gate detection.

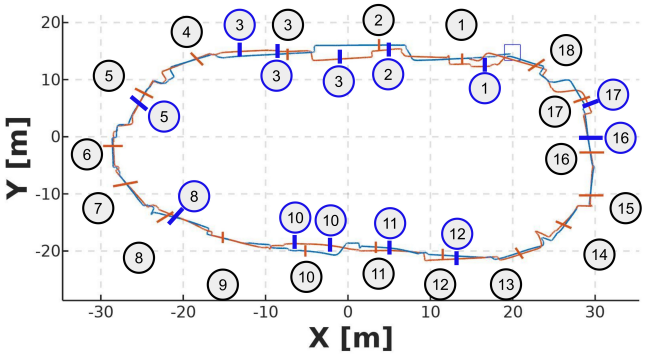


Figure 14: Example of two runs using DeepPilot with automatic gain tuning using gate detection. No changes are made to the gate positions in the light blue trajectory. While in the orange trajectory, the gates are moved along the path. The dark blue circles indicate the number of gates that were moved, and dark blue lines indicate the position to which each of the gates moved. A video illustrating this experiments can be seen at `https://youtu.be/49OzJCHTJu4`

corresponding yaw signal in such a scenario. Therefore, we have proposed to use the gate detector to leverage the performance of DeepPilot by automatically tuning its gains depending upon the gate position on the image. Our experiments in simulation have shown that our proposed approach achieves better results than the original DeepPilot in a large race track with 18 gates randomly placed with variations in orientation and height. For our future work, we will carry out experiments in real scenarios.

References

[1] Hyungpil Moon, Jose Martinez-Carranza, Titus Cieslewski, Matthias Faessler, Davide Falanga, Alessandro Simovic, Davide Scaramuzza, Shuo Li, Michael Ozo, Christophe De Wagter, et al. Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics*, 12(2):137–148, 2019.

[2] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Alphapilot: Autonomous drone racing. *arXiv preprint arXiv:2005.12813*, 2020.

[3] Christophe De Wagter, Federico Paredes-Vallés, Nilay Sheth, and Guido de Croon. The artificial intelligence behind the winning entry to the 2019 ai robotic racing competition. *arXiv preprint arXiv:2109.14985*, 2021.

[4] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. *Science Robotics*, 6(56):eabh1221, 2021.

[5] Rogerio Bonatti, Ratnesh Madaan, Vibhav Vineet, Sebastian Scherer, and Ashish Kapoor. Learning visuomotor policies for aerial navigation using cross-modal representations. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1637–1644. IEEE, 2020.

[6] Leticia Oyuki Rojas-Perez and Jose Martinez-Carranza. Deeppilot: A cnn for autonomous drone racing. *Sensors*, 20(16):4524, 2020.

[7] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A

Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[8] Sunggoo Jung, Hanseob Lee, Sunyou Hwang, and David Hyunchul Shim. Real time embedded system framework for autonomous drone racing using deep learning techniques. In *2018 AIAA Information Systems-AIAA Infotech@ Aerospace*, page 2138. 2018.

[9] J Arturo Cocoma-Ortega, L Oyuki Rojas-Perez, Aldrich A Cabrera-Ponce, and J Martinez-Carranza. Overcoming the blind spot in cnn-based gate detection for autonomous drone racing. In *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, pages 253–259. IEEE, 2019.

[10] Aldrich A Cabrera-Ponce, Leticia Oyuki Rojas-Perez, Jesus Ariel Carrasco-Ochoa, Jose Francisco Martinez-Trinidad, and Jose Martinez-Carranza. Gate detection for micro aerial vehicles using a single shot detector. *IEEE Latin America Transactions*, 17(12):2045–2052, 2019.

[11] S Li, C De Wagter, CC de Visser, QP Chu, GCHE de Croon, et al. In-flight model parameter and state estimation using gradient descent for high-speed flight. *International Journal of Micro Air Vehicles*, 11:1756829319833685, 2019.

[12] Shuo Li, Michal MOI Ozo, Christophe De Wagter, Guido CHE de Croon, et al. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *Robotics and Autonomous Systems*, page 103621, 2020.

[13] Shuo Li, Erik van der Horst, Philipp Duernay, Christophe De Wagter, Guido CHE de Croon, et al. Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone. *Journal of Field Robotics*, 2020.

[14] Shuo Li, Michaël MOI Ozo, Christophe De Wagter, Guido CHE de Croon, et al. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *Robotics and Autonomous Systems*, 133:103621, 2020.

[15] Sunggoo Jung, Sunyou Hwang, Heemin Shin, David Hyunchul Shim, et al. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 3(3):2539–2544, 2018.

[16] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 690–696. IEEE, 2019.

[17] Nitin J Sanket, Chahat Singh, Kanishka Ganguly, Cornelia Fermuller, and Yiannis Aloimonos. Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robotics and Automation Letters*, 2018.

[18] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, Davide Scaramuzza, et al. Deep drone racing: Learning agile flight in dynamic environments. In *Conference on Robot Learning*, pages 133–145. PMLR, 2018.

[19] Jos Arturo Cocoma-Ortega and J Martinez-Carranza. A cnn based drone localisation approach for autonomous drone racing. In *11th International Micro Air Vehicle Competition and Conference*, 2019.

[20] José Arturo Cocoma-Ortega and José Martínez-Carranza. Towards high-speed localisation for autonomous drone racing. In *Mexican International Conference on Artificial Intelligence*, pages 740–751. Springer, 2019.

[21] Christian Pfeiffer and Davide Scaramuzza. Expertise affects drone racing performance. *arXiv preprint arXiv:2109.07307*, 2021.

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[23] H Huang and J Sturm. Tum simulator, 2014.