A Vision-Based Algorithm for a Path Following Problem

Mario Terlizzi[1], Giuseppe Silano[2], Luigi Russo[1], Muhammad Aatif[1], Amin Basiri[1], Valerio Mariani[1], Luigi Iannelli[1], and Luigi Glielmo[1] *†‡

Abstract — **A novel prize-winner algorithm designed for a path following problem within the Unmanned Aerial Vehicle (UAV) field is presented in this paper. The proposed approach exploits the advantages offered by the pure pursuing algorithm to set up an intuitive and simple control framework. A path for a quad-rotor UAV is obtained by using downward facing camera images implementing an Image-Based Visual Servoing (IBVS) approach. Numerical simulations in MATLAB® together with the MathWorks™ Virtual Reality (VR) toolbox demonstrate the validity and the effectiveness of the proposed solution. The code is released as open-source making it possible to go through any part of the system and to replicate the obtained results.**

Index Terms — **Path following, UAV, multi-rotor, virtual reality, image processing.**

1 Introduction

Path following is a relevant application problem within the Unmanned Aerial Vehicle (UAV) field. For instance, in precision agriculture scenarios having good path following algorithms is a fundamental requirement to preserve high productivity rates and plant growth [1]. In civilian applications, monitoring of power lines can be encoded as a path following problem between several target regions that need to be inspected [2]. Whatever the field of application is, drones have to follow a path to accomplish the mission specifications safely and successfully.

Looking at the path following problem, it can be divided into two parts: *detection* and *path following* [3, 4]. As regards the *path detection*, Hough transform [5] and its further developments [6, 7] are considered the most valuable solutions in the literature to cope with the task. However, such a transformation demands a high computational effort making it hard to run on-board the aircraft when battery and processing constraints are tight. These requirements are increasingly stringent when considering Micro Aerial Vehicles (MAVs), where the sensor equipment and the vehicle dimensions are minimal. On the other hand, when the computational burden is at minimum, e.g., lightweight machine learning solutions have been recently proposed to tackle the problem [8, 9], the time required to set up the algorithms makes them difficult to apply in real world applications. For all such reasons, it is of interest to have low computational intensity algorithms, possibly without any prior knowledge of the surrounding environment, able to provide references to the *path following* in a given time window.

Figure 1: Parrot Mambo quad-rotor [18].

Moving at the *path following* level, much of the state-of-the-art solutions [10, 11] rely on the use of nonlinear guidance law [12], vector field [13], and pure pursuit [14] algorithms due to their simple implementation and ease of use. Although the choice of path planner is application sensitive, general considerations can be provided. The performance of nonlinear guidance law degrades as the target acceleration changes rapidly introducing a not negligible delay in the trajectory generation. An adequate knowledge of the target velocity and acceleration is required to avoid instability issues [12]. On the other hand, a vector field solution prevents from such oscillations problems, but it is inherently characterized by a high computational effort [13]. Besides, a pure pursuit approach is a suitable solution when tracking error and computational effort are critical. The position of a look-ahead point is set up at the beginning of the mission, and then updated at each step following some tracking criteria [15–17]. The objective is to reduce the distance between the current position and the look-ahead position.

In this paper, we propose a novel winner-prize algorithm designed to deal with the path following problem in the context of the IFAC2020 MathWorks Minidrone competition [19]. The framework combines the advantages provided by the pure pursuit algorithm and some simple image processing to detect and to track a pre-established path. The lightweight and ease of implementation of the proposed solution allow the deployment on low computational capacity MAVs, such as the Parrot Mambo [18] (see, Figure 1) considered as a testbed for the application. Numerical simulations carried out in MATLAB together with the MathWorks Virtual Reality (VR) toolbox [20] show the validity and the effectiveness of the proposed approach. Moreover, the code is provided as open-source [21] making it possible

*This project was partially funded by the ECSEL Joint Undertaking (JU) research and innovation programme AFarCloud and COMP4DRONES under grant agreement no. 783221 and no. 826610, respectively, and by the European Union's Horizon 2020 research and innovation programme AERIAL-CORE under grant agreement no. 871479. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, Austria, Portugal, Sweden, Finland, Czech Republic, Poland, Italy, Latvia, Greece, and Norway.

†1 Mario Terlizzi, Luigi Russo, Muhammad Aatif, Amin Basiri, Valerio Mariani, Luigi Iannelli, and Luigi Glielmo are with the Department of Engineering, University of Sannio in Benevento, Benevento, Italy (email: {mterlizzi, luirusso, maatif, basiri, vmariani, luiannel, glielmo}@unisannio.it).

‡2 Giuseppe Silano is with the Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic (email: giuseppe.silano@fel.cvut.cz)

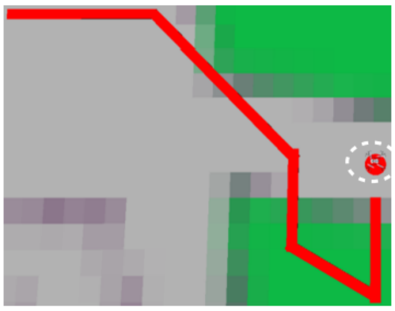

Figure 2: Snapshot extracted from the virtual scenario. A dashed circle is used to indicated the drone position.

to go through any part of system and to replicate the obtained results.

The paper is organized as follows. Section 2 presents the problem, while in Sec. 3 the vision-based path following algorithm is described. Numerical simulations are reported in Sec. 4. Finally, conclusions are drawn in Sec. 5.

2 Problem Description

The work presented here finds an application within the IFAC2020 MathWorks Minidrone competition [19], where the use of a model-based design approach is the aim of the specific contest. Path error and mission time are used as evaluation metrics for the algorithm. The whole process is the following: a quad-rotor UAV follows a pre-established red path by using its downward facing camera to get feedback from the environment. Images are updated according to the position and orientation of the vehicle simulated in the MATLAB VR world. No prior knowledge on the path and the surrounding scenario is given. The drone takes off and starts its motion looking at the path, and the mission stops with the recognition of an end-marker. At that time, the drone lands staying within the delimited area. Figure 2 shows the considered scenario.

3 Vision-Based Path Following Algorithm

The vision-based path following algorithm combines the advantages offered by the pure pursuit algorithm [22] with that of an easy image processing system to cope with the task. The algorithm starts selecting a target position ahead of the vehicle and that has to be reached, typically on a path. The framework is based on the following operations: (i) given the actual position $\mathbf{d} = (x_d, y_d)^\top \in \mathbb{R}^2$ where the UAV is located, a Virtual Target Point (VTP) is set over the track at $\mathbf{w} = (x_w, y_w)^\top \in \mathbb{R}^2$; then, (ii) the quad-rotor is commanded to reach the VTP along a straight line (essentially it is the pure pursuit algorithm with a curvature of infinite radius) [22], i.e., moving the vehicle from its current position to the goal position[1]. In Figure 3 an illustrative example of how the algorithm works is depicted.

Contrary to the pure pursuit algorithm, the proposed approach exploits the intrinsic characteristics of the multi-rotor

[1]The quad-rotor is assumed to fly at a fixed height along the entire mission.

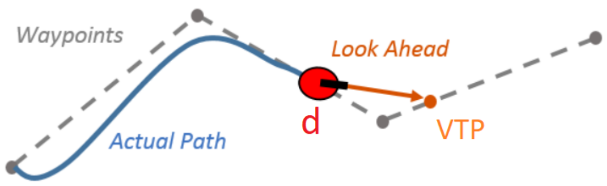

Figure 3: An illustrative example of the proposed vision-based path following algorithm works. The red point $\mathbf{d}$ represents the drone position, while the orange point $\mathbf{w}$ depicts the VTP.

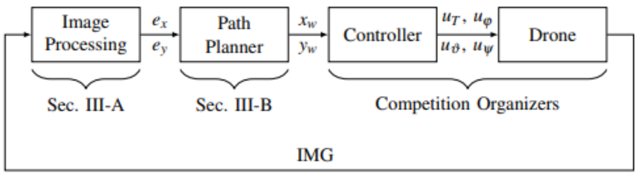

Figure 4: Control system architecture. From left to right: the image processing, path planner, controller, and drone blocks.

UAV motion: differently from ground vehicles with steering wheels, drones can follow a path without modifying their heading. Such an assumption allows reducing the time to accomplish the task by removing the control of the heading from the purposes of the path follower.

In Figure 4 the whole control scheme architecture is reported. The algorithm is mainly divided into two parts: (i) the Image Processing System (IPS) deals with extracting the red path from the camera images, providing the errors along the x- and y-axis of the camera frame between the current drone position and the VTP point, and recognizing the End-Marker for the landing operations; while, (ii) the Path Planner (PP) figures out the path following problem by computing the new position $\mathbf{w}$ of the drone in the world frame [20, Sec. V] implementing an Image-Based Visual Servoing (IBVS) scheme. The control algorithm computes the commands u_T, u_φ, u_ϑ, and u_ψ that should be given to the drone in order to update its position and orientation in accordance to the PP references.

The overall mission is divided into four parts: *Take off*, *Following*, *End-Marker*, and *Landing*. A decision-making process has been implemented to achieve the competition objectives triggering the system from a state to another, as depicted in Figure 5. For each frame, the IPS accesses the system status and plan the next action (i.e., landing, following, etc.). The drone starts taking off from its initial position looking at the path. Once the vehicle reaches the hovering position, the IPS detects the path and the state machine enters in the *Following* state, hence the path following starts. As soon as the IPS detects the End-Marker, the state machine exits from the *Following* state and goes into the *End-Marker* state. At this stage the mission stops, and the drone starts the landing. In the following subsections the implementation of the image processing system and path planner modules are detailed.

3.1 Image processing system

Starting from the camera frames, the Image Processing System takes care of separating the features of the pre-

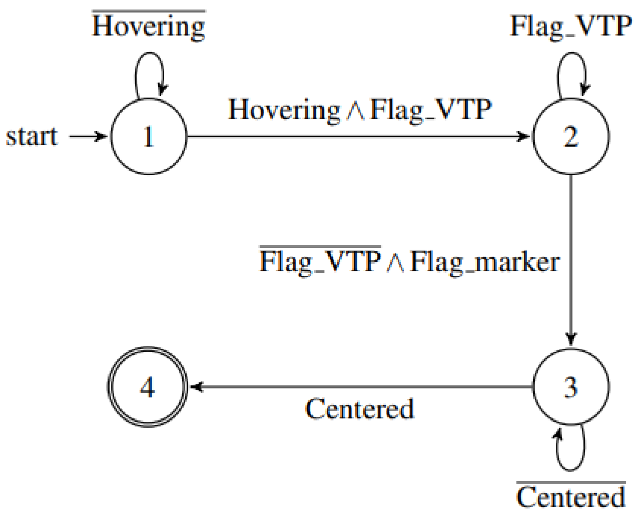
$\overline{\text{Hovering}}$
Flag_VTP
start
1
Hovering ∧ Flag_VTP
2
$\overline{\text{Flag\_VTP}}$ ∧ Flag_marker
4
Centered
3
$\overline{\text{Centered}}$

Figure 5: State machine implemented. State 1: *Take-off*. State 2: *Following*. State 3: *End-Marker*. State 4: *Landing*.

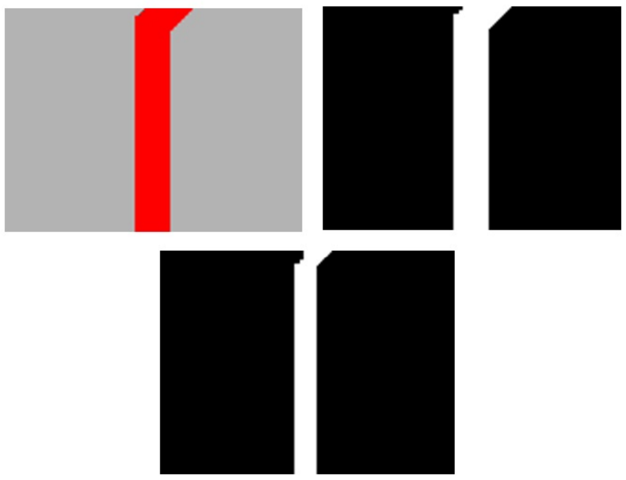

Figure 6: Original frame (upper left), converted and binarized frame (upper right), and eroded frame (lower).

established path from that of the environment. The IPS receives frames of width W and height H from the camera sensor at each $T_{\text{IPS}} = 0.2\text{s}$, i.e., the camera sampling time. The image format is RGB with 8 bits for each color channel. The path is 0.01 m in width, while the landing marker is circular with a diameter of 0.02 m. The path color is red, and this information is taken into consideration in all the elaborations to filter out the background scenario. The procedure consists of the following steps: first, the RGB frame is converted into an intensity level frame representation as follows

$$F(n,m) = f_R(n,m) - \frac{f_G(n,m)}{GG} - \frac{f_B(n,m)}{GB}, \qquad (1)$$

where the pair (n,m) represents the pixel located at row $n \in \{1,2,\ldots,H\}$ and column $m \in \{1,2,\ldots,W\}$ of the image frame and f_i, with $i \in \{R,G,B\}$, provides the intensity level representation of the corresponding red, green and blue channels. An heuristic approach was used to tune the GG, $GB \geq 1$ parameter values. These parameters help to detect the pixels belonging to the path. Further, a binarization process based on a K_T threshold value refines the process removing artifacts from the elaboration. The binarized frame can be described by the binary function $F_{\text{bin}}\colon (n,m) \to \{0,1\}$ whose output is one when the pixel belongs to the path and zero otherwise. Finally, an erosion operation is performed through a square kernel to shrink and regularize the binarized frame. In Figure 6 the overall process is reported for a single sample frame.

Then, the obtained reference path is used in a twofold way: (i) to identify a new VTP belonging to the track; (ii) to detect the landing marker. The two tasks are described in the pseudocode reported in Algorithm 1.

Looking at the algorithm, the first three functions (i.e., `channelConv`, `binarization`, and `erosion`) take care of extracting the path information from the frame. Then, the `detectTrack` and `detectMarker` functions deal with raising a flag when the path (`Flag_VTP`) or the End-Marker (`Flag_marker`) are detected. The path following algorithm starts with the IPS that computes the errors (e_x and e_y) between the drone position and the VTP point for the PP by using a circular arc mask centered in the drone Center of Mass (CoM)[2] with thickness $R_{\max} - R_{\min}$ [3].

Algorithm 1: Image Processing System

```
1  IMG ← channelConv(IMG),
2  IMG ← binarization(IMG),
3  IMG ← erosion(IMG),
4  Flag_VTP ← detectTrack(IMG),
5  Flag_marker ← detectMarker(IMG)
6  if Flag_VTP then
7          $x_{\text{VTP}}, y_{\text{VTP}}$ ← vtp(frame)
8          $e_x \leftarrow x_{\text{VTP}} - x_{\text{CoM}}$
9          $e_y \leftarrow y_{\text{VTP}} - y_{\text{CoM}}$
10 else
11     if Flag_marker then
12             $x_{\text{MARK}}, y_{\text{MARK}}$ ← cgMarker(frame)
13             $e_x \leftarrow x_{\text{MARK}} - x_{\text{CoM}}$
14             $e_y \leftarrow y_{\text{MARK}} - y_{\text{CoM}}$
15 return $e_x$, $e_y$, Flag_VTP, Flag_marker
```

In Figure 7, the arc mask considering the VTP position at time $\mathbf{t}_k$ is depicted, where $\mathbf{t}_k$ denotes the k-element of the time interval vector defined as $\mathbf{t} = (0, T_{\text{IPS}}, \ldots, NT_{\text{IPS}})^\top \in \mathbb{R}^{N+1}$, with $k \in \mathbb{N}_0$. The orientation angle $\vartheta = \text{arctan2}(x_{\text{VTP}}, y_{\text{VTP}})$ is calculated with respect to the frame coordinates, where the arctan2 function is the four-quadrant inverse of the tangent function. A portion $\varTheta$ of the arc mask is established by taking into account the previous VTP's orientation. In particular, we set up two semi-arcs with width $\frac{\varTheta}{2}$, namely Field of View (FoV), in counter-clockwise and clockwise directions from ϑ. Then, the arc mask is applied to the eroded image obtaining the VTP point at $\mathbf{t}_{k+1}$. The function TP calculates x_{VTP}, y_{VTP}, and ϑ which represent the frame coordinates and angle orientation of the VTP at $\mathbf{t}_{k+1}$, respectively. Subsequently the corresponding errors with respect to the center of mass, i.e., e_x and e_y, are computed inside the frame coordinates. Finally, the `Flag_VTP` and the e_x and e_y values are provided

[2]The assumption that the CoM being in the center of the reference frame, i.e., $x_{\text{CoM}} = \frac{H}{2}$ and $y_{\text{CoM}} = \frac{W}{2}$, is taken into consideration.
[3]$R_{\max}$ and $R_{\min}$ are the outer and inner radius, respectively.

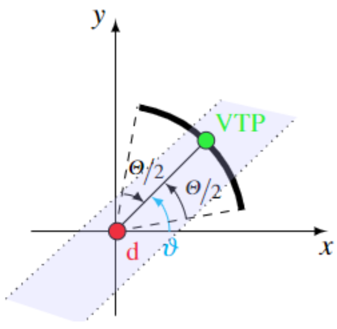

Figure 7: Arc mask. The drone position (red), the previous VTP (green), and the pre-established path to follow (purple) are reported.

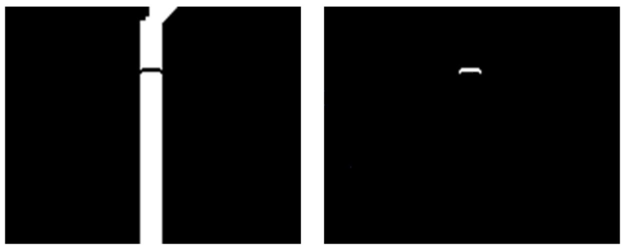

Figure 8: Frame after the application of the Arc mask (left). Extracted pixels belonging to the path (right).

as input to the PP at each T_{IPS}. Figure 8 shows the result of the entire process setup.

It is worth noticing that when the landing marker is detected and no other VTP point is found in the frame, the IPS triggers the state machine in the `End-Marker` state. Here, the new main task of the IPS is to obtain the position of the End-Marker within the frame coordinates. An additional erosion process is performed by using a circular kernel, as depicted in Figure 9.

3.2 Path planner

The Path Planner is designed to compute the position of the VTP point $\mathbf{w} = (x_w, y_w)^\top$ maintaining a constant altitude (z_H) while following the path. Roughly speaking, the PP computes the spatial coordinates x_w and y_w trying to reduce the errors, i.e., e_x and e_y, between the drone position and the VTP. These values are later used by the drone controller to tune the command signals u_T, u_φ, u_ϑ, and u_ψ, as described in Figure 4. The proposed path planner is based on Proportial-Integra control loops. As a common rule in cascade structure, the inner loop, i.e., the PP, is regulated at a rate faster than the outer loop, i.e., the IPS. In our case, the PP runs at 200 Hz ($T_{\text{PP}} = 5X10^{-3}$s) while the IPS runs at 2 Hz ($T_{\text{IPS}} = 0.2$s). These are a standard solution in the literature for quad-rotors control design [23].

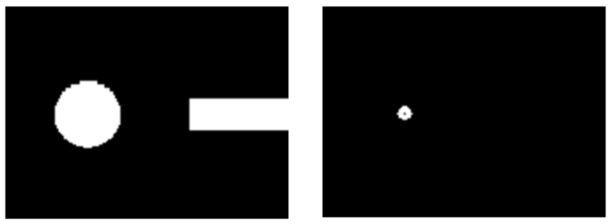

Figure 9: Original (left) and eroded frames (right) of the End-Marker are reported.

As described in Sec. 3, the path following stops with the detection of the End-Marker. At that time, the IPS implements a toggle switch behavior raising the `Flag_marker` flag while holding low the `Flag_VTP` flag. This mutually separates the *Following* and *Landing* phases avoiding instability issues. The pseudocode of the proposed algorithm is reported in Algorithm 2 with parameter values detailed in Table 1.

In Appendix 5, we show how α can be set to control the velocity of vehicle along the entire mission. Therefore, the proposed Vision-Based Path Following algorithm makes it possible not only to generate the spatial coordinates x_w and y_w using a IBVS scheme but also to set the velocity during the entire mission.

Algorithm 2: Path Planner

```
1  $e_x$, $e_y$, Flag_VTP, Flag_marker
2  if Flag_VTP then
3      $x_{k+1} \leftarrow x_k + \alpha e_x$
4      $y_{k+1} \leftarrow y_k + \alpha e_y$
5      $z_{k+1} \leftarrow z_H$
6  if Flag_marker then
7      if $(e_x = 0 \wedge e_y = 0)$ then
8          $x_{k+1} \leftarrow x_k$
9          $y_{k+1} \leftarrow y_k$
10         $z_{k+1} \leftarrow 0$
11     else
12         $x_{k+1} \leftarrow x_k + \beta e_x$
13         $y_{k+1} \leftarrow y_k + \beta e_y$
14         $z_{k+1} \leftarrow z_H$
15 $x_w \leftarrow x_{k+1}$, $y_w \leftarrow y_{k+1}$, $z_w \leftarrow z_{k+1}$
16 return $x_w$, $y_w$, $z_w$
```

4 Numerical Results

To demonstrate the validity and effectiveness of the proposed framework, numerical simulations have been carried out by using the 2019b release of MATLAB equipped with MathWorks Virtual Reality toolbox [24] and Parrot support package for Simulink [18]. The video available at [25] illustrates in a direct way how the system works, i.e., the ability of the quad-rotor UAV to follow the pre-established red path and to land on the End-Marker. In addition, the video shows the behavior of the IPS and PP that never lose the path during the entire mission.

In Figure 10 a comparison of the system performance by using various values of α is reported. As can be seen from the plots, the larger α is, the lower the mission time (T_s) is. On the other hand, the lower the mission time is, the greater the path error is. Looking at the zoom plot (see, Figure 10) it is even clearer how the system performance degrades with increasing α value, and these are all the more evident as the path is angular. For the considered scenario, an heuristic approach was used to tune the α and β parameter values.

Table 1: Parameter values.

	Sym	Value
Sampling time	T_{PP}	5×10^{-3} s
Sampling time	T_{IPS}	0.2s
PP constant	β	18×10^{-3} pixel^{-1}
Frame height	H	120 pixel
Frame width	W	160 pixel
Outer radius Arc mask	$R_{\max}$	28 pixel
Inner radius Arc mask	$R_{\min}$	26 pixel
IPS constant	GB	2
IPS constant	GG	2
IPS threshold	K_T	150
FOV Arc mask	ϑ	2.3 rad
Drone height	z_H	1 m

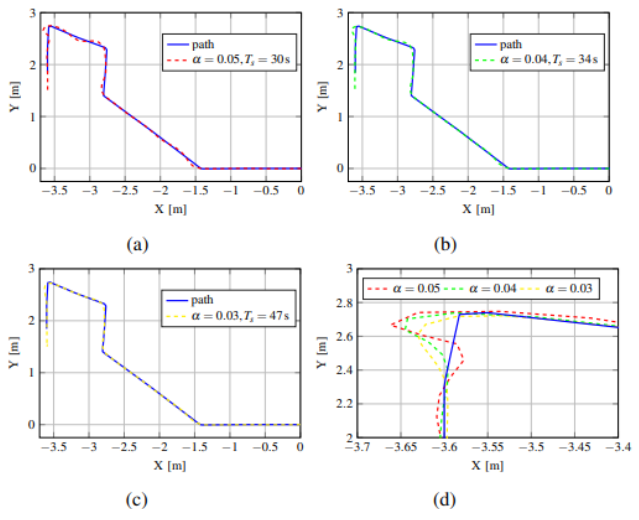

Figure 10: Trajectory plots. From left to right: the desired and the drone paths for various values of α are represented. The mission time T_s and a comparison between the considered α values are also reported.

Figure 11 depicts the drone velocity v_x and v_y along the x- and y-axis, respectively, and the norm of the drone velocity v_D. As described in Sec. 3.2 and detailed in Appendix 5, the norm of the drone velocity remains approximately constant while following the path. The presence of spikes might be due to the coupling effects of the drone xy dynamics even though x_w and y_w references have not been modified yet (see, Figure 10). Such coupling effects are probably caused by the asymmetric positioning of the rotors with respect to the principal axis and the effect of the discrete image pixelization.

5 Conclusion

In this paper, a prize-winner algorithm designed for a path following problem within the IFAC2020 Mathworks Minidrone Competition has been presented. In particular, a lightweight and easy of implementation solution was set to generate the spatial coordinates of the VTP point to fulfill the competition requirements successfully. Numerical simulations carried out in MATLAB together with the MathWorks Virtual Reality toolbox and the Parrot support package for Simulink demonstrated the validity and the effectiveness of the proposed approach. The software has been released as open-source making it possible to go through any part of the

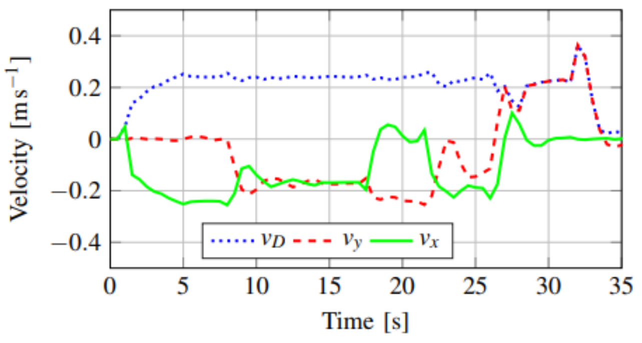

Figure 11: Velocity plot.

system and to replicate the obtained results. Future work will include the integration of more challenging features, such as obstacle avoidance and 3-D reference generation, and lead to field experiments. Furthermore, the effects of the drone dynamics on the performance of the vision-based system will be also explored.

References

[1] B. Maik and d. E. Pignaton, "A UAV guidance system using crop row detection and line follower algorithms," *Journal of Intelligent & Robotic Systems*, pp. 1–17, 2019.

[2] G. Silano, T. Baca, R. Penicka, D. Liuzza, and M. Saska, "Power Line Inspection Tasks with Multi-Aerial Robot Systems via Signal Temporal Logic Specifications," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 4169–4176, 2021.

[3] B. Dahroug, J.-A. Séon, A. Oulmas, T. Xu, B. Tamadazte, N. Andreff, and S. Régnier, "Some Examples of Path Following in Microrobotics," in *International Conference on Manipulation, Automation and Robotics at Small Scales*, 2018, pp. 1–6.

[4] M. A. Rafique and A. F. Lynch, "Output-Feedback Image-Based Visual Servoing for Multirotor Unmanned Aerial Vehicle Line Following," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 4, pp. 3182–3196, 2020.

[5] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[6] D. Dagao, X. Meng, M. Qian, H. Zhongming, and W. Yueliang, "An improved Hough transform for line detection," in *2010 International Conference on Computer Application and System Modeling*, vol. 2, 2010, pp. V2–354.

[7] S. Du, B. J. van Wyk, C. Tu, and X. Zhang, "An Improved Hough Transform Neighborhood Map for Straight Line Segments," *IEEE Transactions on Image Processing*, vol. 19, no. 3, pp. 573–585, 2010.

[8] V. Nhan, J. Robert, and R. Davide, "LS-Net: Fast Single-Shot Line-Segment Detector," *Machine Vision and Applications*, vol. 12, no. 32, 2020.

[9] J. Tang, S. Li, and P. Liu, "A review of lane detection methods based on deep learning," *Patter Recognition*, vol. 111, pp. 1–15, 2021.

[10] P. Sujit, S. Saripalli, and J. B. Sousa, "An evaluation of UAV path following algorithms," in *2013 European Control Conference*, 2013, pp. 3332–3337.

[11] G. V. Pelizer, N. B. Da Silva, and K. R. Branco, "Comparison of 3d path-following algorithms for unmanned aerial vehicles," in *2017 International Conference on Unmanned Aircraft Systems*, 2017, pp. 498–505.

[12] S. Keshmiri, A. R. Kim, D. Shukla, A. Blevins, and M. Ewing, "Flight Test Validation of Collision and Obstacle Avoidance in Fixed-Wing UASs with High Speeds Using Morphing Potential Field," in *2018 International Conference on Unmanned Aircraft Systems*, 2018, pp. 589–598.

[13] T. Tuttle, T. T. Moleski, and J. Wilhelm, "Multi-Rotor Path-following Performance using Vector Field Guidance and Velocity Control," in *AIAA Scitech 2021 Forum*, 2021.

[14] A. S. Baqir and A. A. Ammar, "Navigation of Mini Unmanned Aerial Vehicle in Unknown Environment," *IOP Conference Series: Materials Science and Engineering*, vol. 745, 2020.

[15] A. Gautam, P. B. Sujit, and S. Saripalli, "Application of guidance laws to quadrotor landing," in *2015 International Conference on Unmanned Aircraft Systems*, 2015, pp. 372–379.

[16] D. M. Xavier, B. F. S. Natassya, and R. Branco Kalinka, "Path-following algorithms comparison using Software-in-the-Loop simulations for UAVs," in *2019 IEEE Symposium on Computers and Communications*, 2019, pp. 1216–1221.

[17] G. Silano, P. Oppido, and L. Iannelli, "Software-in-the-loop simulation for improving flight control system design: a quadrotor case study," in *IEEE International Conference on Systems, Man and Cybernetics*, 2019, pp. 466–471.

[18] MathWorks, "Simulink Support Package for Parrot Minidrones." [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/63318-simulink-support-package-for-parrot-minidrones

[19] Mathworks, *Mathworks Minidrone Competition IFAC20*. [Online]. Available: https://it.mathworks.com/academia/student-competitions/minidrones/ifac-2020.html

[20] G. Silano and L. Iannelli, "MAT-Fly: An Educational Platform for Simulating Unmanned Aerial Vehicles Aimed to Detect and Track Moving Objects," *IEEE Access*, vol. 9, pp. 39 333–39 343, 2021.

[21] M. Terlizzi, "Vision Based Pure Pursuing Algorithm," GitHub repository. [Online]. Available: https://github.com/mar4945/Vision-Based-Pure-Pursuing-Algorithm

[22] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992. [Online]. Available: http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/MRIS/coulter_r_craig_1992_1.pdf

[23] T. N. Dief and S. Yoshida, "Review: Modeling and Classical Controller Of Quad-rotor," *International Journal of Computer Science and Information Technology & Security*, vol. 5, no. 4, pp. 314–319, 2015.

[24] MathWorks, "Simulink 3D Animation toolbox." [Online]. Available: https://www.mathworks.com/products/3d-animation.html

[25] M. Terlizzi, "MATLAB Minidrone Competition IFAC20," YouTube. [Online]. Available: https://youtu.be/9VySp0j-1hc

Appendix I

Let us consider a continuous-time dynamical system $\mathcal{H}$ and its discrete time version $x_{k+1} = f(x_k, u_k)$, where $x_k, x_{k+1} \in X \subset \mathbb{R}^n$ are the current state and the next state of the system, respectively, $u \in U \subset \mathbb{R}^m$ is the control input. Let us consider the PP algorithm implementation detailed in Algorithm 2. Hence, the next state of the system x_{k+1} and y_{k+1} along the x- and y-axis can be written as follows:

$$x_{k+1} = x_k + \alpha e_{x_k}, \qquad y_{k+1} = y_k + \alpha e_{y_k}, \tag{2}$$

respectively. After some simple algebra, we can write:

$$\frac{x_{k+1} - x_k}{T_{\mathrm{PP}}} = \frac{\alpha e_{x_k}}{T_{\mathrm{PP}}}, \qquad \frac{y_{k+1} - y_k}{T_{\mathrm{PP}}} = \frac{\alpha e_{y_k}}{T_{\mathrm{PP}}}, \tag{3}$$

and hence,

$$v_x \approx \frac{\alpha e_{x_k}}{T_{\mathrm{PP}}} = \tilde{\alpha} e_{x_k}, \qquad v_y \approx \frac{\alpha e_{y_k}}{T_{\mathrm{PP}}} = \tilde{\alpha} e_{y_k}, \tag{4}$$

with $\tilde{\alpha} = \frac{\alpha}{T_{\mathrm{PP}}}$.

Knowing that e_{x_k} and e_{y_k} are by definition the projections over a circle along the x- and y-axis of the VTP with an angle ϑ_k, we can write

$$\begin{aligned} e_{x_k} &= \frac{R_{\max} + R_{\min}}{2} \sin \vartheta_k, \\ e_{y_k} &= \frac{R_{\max} + R_{\min}}{2} \cos \vartheta_k, \end{aligned} \tag{5}$$

and thus,

$$V_D = \sqrt{v_x^2 + v_y^2} \approx \tilde{\alpha} \frac{R_{\max} + R_{\min}}{2}. \tag{6}$$

Hence the parameter α controls the drone velocity. ∎