Journal of Intelligent & Robotic Systems (2022) 105: 17
https://doi.org/10.1007/s10846-022-01612-5

REGULAR PAPER

Ornithopter Trajectory Optimization with Neural Networks and Random Forest

M. A. Pérez-Cutiño[1,2] · F. Rodríguez[1] · L. D. Pascual[3] · J. M. Díaz-Báñez[1]

Received: 1 October 2021 / Accepted: 5 March 2022 / Published online: 7 May 2022

Abstract
Trajectory optimization has recently been addressed to compute energy-efficient routes for ornithopter navigation, but its online application remains a challenge. To overcome the high computation time of traditional approaches, this paper proposes algorithms that recursively generate trajectories based on the output of neural networks and random forest. To this end, we create a large data set composed by energy-efficient trajectories obtained by running a competitive planner. To the best of our knowledge our proposed data set is the first one with a high number of pseudo-optimal paths for ornithopter trajectory optimization. We compare the performance of three methods to compute low-cost trajectories: two classification approaches to learn maneuvers and an alternative regression method that predicts new states. The algorithms are tested in several scenarios, including the landing case. The effectiveness and efficiency of the proposed algorithms are demonstrated through simulation, which show that the machine learning techniques can be used to compute the flight path of the ornithopter in real time, even under uncertainties such as wrong sensor readings or re-positioning of the target. Random Forest obtains the higher performance with more than 99% and 97% of accuracy in a landing and a mid-range scenario, respectively.

Keywords Trajectory optimization · Neural networks · Random forest · Ornithopter · Dataset

Multimedia Material

The Ornithopter Trajectory Optimization (OTO) data set and evaluation code can be found at https://github.com/mpcutino/OTO_dataset.

1 Introduction

An ornithopter is a flapping wing drone [5] and it is normally designed to imitate the flight of a bird, a bat or an insect. These prototypes seek to overcome the battery limitation and the human-safety problems of the multi-rotor drones. Over the past few decades, there have been remarkable advances in the design of flapping wing vehicles [23, 31]. However, autonomous navigation remains a challenging task, among other reasons, due to lack of adequate motion planning algorithms. In robotics, motion planning is the problem of computing a path that connects an initial and a target robot configuration, also known as a trajectory. Motion planning for avoiding collisions has been the most common goal [12], but other objectives such as smooth paths or low battery consumption can be defined.

Trajectory optimization algorithms aim to find the best trajectory for an object in a space when a set of kinematics and dynamics rules are considered [13]. An optimal trajectory is the one that minimizes (or maximizes) a given cost function and satisfies a set of constraints, usually described in a mathematical differential model. The complexity of computing an optimal trajectory is closely related to the number of maneuvers that the object can perform at each point of the trajectory (i.e. the number of paths that can be generated). Thus, planning optimal trajectories for autonomous ornithopters is a complicated problem. First,

✉ M. A. Pérez-Cutiño
mpcutino@us.es;m.perez@virtualmech.com

Extended author information available on the last page of the article.

these types of vehicles present nonlinear and complex dynamics which must be taken into account when computing feasible trajectories. Second, the state space is large as it must include the position, velocities and altitude of the drone, which are relevant for gliding and perching operations. Due to these complexities, there is a need for efficient model-based methods that allow the computation of pseudo-optimal trajectories with real-time performance. To address this gap in the research, in this paper we propose supervised Machine Learning (ML) algorithms to learn to fly an ornithopter. Due to the lack of data from real bird flights, our idea is to generate artificial bio-inspired flight data sets consisting of pseudo-optimal trajectories computed by a competitive planner.

A recent approach for the problem of optimizing ornithopters trajectories is [27]. The authors proposed the planner OSPA (which stands for Ornithopter Segmentation-based Planning Approach), a trajectory optimization algorithm for a real ornithopter prototype. This algorithm can be applied to any dynamic model and different flight types. The authors demonstrated that OSPA outperforms alternative probabilistic kinodynamic planners both in cost (total energy) and accuracy (distance to the target). Unfortunately, OSPA is only valid for online application in short distances scenarios. Furthermore, in these scenarios OSPA only minimizes the positional distance, without taking into account the ornithopter velocity and pitch values at the target configuration.

Here, we present new approaches to plan online energy-efficient trajectories for an ornithopter, both for long and short distances scenarios. The goal is improving the efficiency of the algorithms without degrading the quality of its solutions. Our method uses neural networks (NNs) and Random Forest (RF) to significantly improve the computational time of the OSPA planner. To train the supervised methods, we build a data set of energy-optimized trajectories based on the OSPA algorithm; see Fig. 1(a). We design ML algorithms that can learn an optimal policy from the data set, and predict the sequence of states of an energy-efficient trajectory that considers transition between flapping and gliding, in order to save as much energy as possible, as illustrated with the example of Fig. 1(b)).

In our previous work [25], we provide the first data set containing energy-efficient trajectories for ornithopter flights. In this paper, four main additional extensions are developed: 1) we increase the existing data set with a large number of trajectories, considering a higher number of maneuvers, as detailed in Section 4; 2) we incorporate an efficient Random Forest to accurately predict the next maneuver; 3) we add complex strategies to the networks architecture and the training process, such as residual connections [8], and a weighted loss function to consider class imbalance; 4) we study the performance of the proposed ML algorithms in more complex scenarios, by introducing random noise during flights and forcing re-planning strategies.

In general, our main contributions are the following:

- We create a novel data set for ornithopter trajectory optimization termed as OTO. To the best of our knowledge, this is the first dataset with more than 100 000 pseudo-optimal trajectories for ornithopter motion planning, surpassing by a wide margin the previous existing one (containing less than 1 000 trajectories). The data set is benchmarked with several algorithms, and we provide a training-test split to ensure fair comparisons.
- We design two classifiers: an artificial neural network (ANN) and a Random Forest. The algorithms learn maneuvers and can be queried to predict energy-optimized trajectories that can be computed online. These algorithms provide a strong benckmark for our novel data set.
- We use an alternative recurrent neural network (RNN) to learn states instead of maneuvers, avoiding the need of knowing the model of the drone dynamics.
- We show that the proposed ML-based algorithms can be used for real-time trajectory optimization, both for medium and short distances. Indeed, our methods significantly reduce computational costs while resulting in trajectories comparable to those produced by OSPA.

The remainder of the paper is organized as follows: Section 2 outlines related articles and Section 3 states the problem description. Afterwards, the created data set is introduced in Section 4. The ML-based algorithms are presented in Section 5, while computational analysis and results take place in Section 6. Finally, conclusions are outlined in Section 7.

2 Related Work

Trajectory optimization is a crucial task in autonomous navigation. Over the past decade, the optimization problem has been vastly studied, specially for multirotors and fixed-wing vehicles; see [4] for a comprehensive survey. However, optimizing trajectories with real-time algorithms remains a challenge, particularly for systems with complex dynamics.

Among the existing techniques for trajectory optimization [3], we are interested on developing efficient ML algorithms capable of overcome the existing limitations of traditional methods. In the existing literature, some types of path planners based on NNs have been proposed for Unmanned Aerial Vehicles (UAVs). Most of the methods utilize NNs to approximate the system dynamics, objective functions,

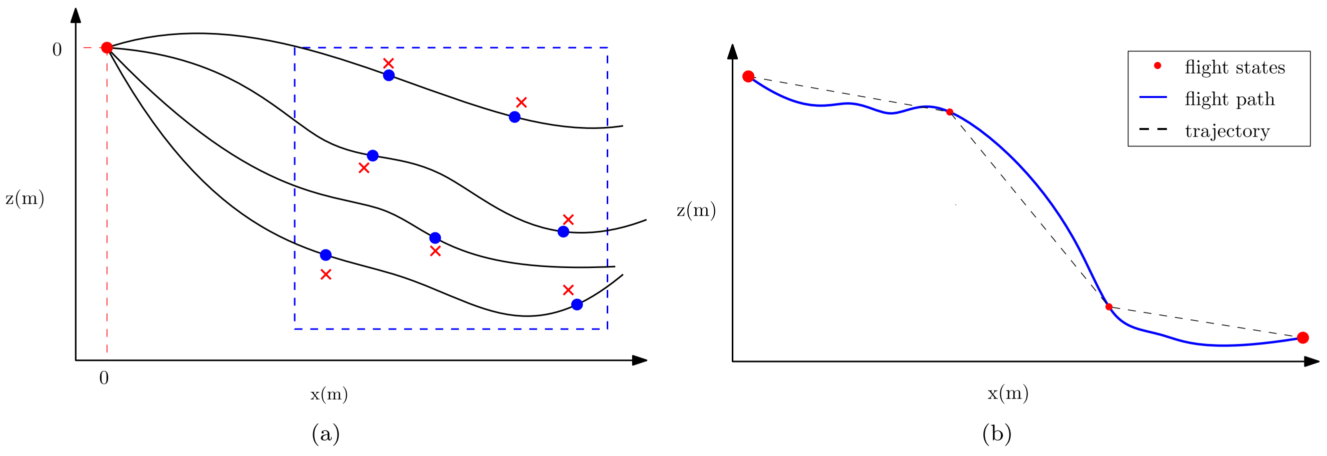


Fig. 1 The proposed ML algorithms learn from experience by observing optimal trajectories starting at a given state (red dot) and targeting a point in space (red crosses), as in Figure (a). Data is obtained by implementing OSPA [27], which generates pseudo-optimal trajectories ending (blue dots) near the target states (red stars). Figure (b) is an example of a desired trajectory with three maneuvers from 15 meters before landing. The landing operation is achieved by a first maneuver with flapping, a second one performing gliding, and the last one using tail deflection

and gradients, which removes the requirement for collocation [10, 15]. Another promising approach is the use of deep NNs. For instance, in the paper [16], a Convolutional NN (CNN) model has been introduced to formulate path planning as a supervised classification problem. On the other hand, recurrent NNs (RNNs) has been proposed for solving 2-D maze navigation in [11], and algorithms combining RNNs with other learning techniques have been recently studied in [7, 32]. Finally, other learning approaches, as Q-learning [34], cooperative learning [37], random forest [22] and reinforcement learning algorithms [38] have been proposed for UAV path planning.

The aforementioned algorithms have been designed for the problem of path planning, which is usually the first step before obtaining an optimal trajectory. Our proposal is capable of generating an optimal trajectory without using an initial trajectory as reference.

Trajectory optimization algorithms using neural networks have been proposed in [17, 18]. In [18], an student-teacher strategy is used to couple an Artificial Neural Network and a trajectory optimization method based on contact-invariant optimization [19]. On the other hand, [17] trained a Multilayer Perceptron on trajectories obtained with the use of differential game theory. Both methods evidences the benefits of using NNs as part of the optimization process. Moreover, when properly controlling the parameters of these algorithms, the NNs can produce real-time responses.

Despite the development of the aforementioned works, trajectory optimization using ML algorithms for complex systems, such as ornithopters, is an under researched area. In this paper, a dataset with a large number of pseudo-optimal trajectories is provided, and several ML-based methods are proposed to achieve ornithopter real-time path planning. The algorithms are tested in complex environments, addressing a trade-off between path quality and efficiency.

3 Problem Description

Suppose that we have an autonomous ornithopter with a known model of its dynamics (see [27] for more details about the specific dynamic model assumed in this paper). The problem is to plan online energy-efficient trajectories to navigate the ornithopter from a starting point to a target location. Those trajectories have to comply with the ornithopter dynamics, therefore they are flyable. Then, we assume the existence of lower-level algorithms to control the ornithopter through the computed trajectory.

A trajectory is a sequence of flight states, where a state describes the ornithopter configuration at a given instant of time. A flight state is given by a tuple $s = (x, z, u, w, \Theta, q)$, where x and z are the positional values, u and w are velocity components in the body reference frame, Θ is the pitch angle and q is the pitch angular velocity. Our trajectories are constrained to the XZ plane, as we assume a longitudinal motion model for the ornithopter.

The ornithopter reaches a state (dynamically feasible) performing a specific control maneuver. A control maneuver is a tuple of values $m = (\delta, f)$, where δ is the tail deflection, determined by the deflection angle (up and down), and f is the wing flapping, determined by the flapping frequency (including zero value for gliding). Thus an energy-efficient trajectory can be described by combining flapping, gliding and tail flick. An example of an ornithopter trajectory is illustrated in Fig. 1(b). In this paper, we establish a maneuver set M composed by all combinations of flapping frequencies

Table 1 The considered maneuvers with their corresponding labels

class	maneuver	class	maneuver
0	$(-4°, 0\ H/z)$	13	$(-2°, 3\ H/z)$
1	$(-4°, 1\ H/z)$	14	$(-2°, 4\ H/z)$
2	$(-4°, 2\ H/z)$	15	$(-1°, 0\ H/z)$
3	$(-4°, 3\ H/z)$	16	$(-1°, 1\ H/z)$
4	$(-4°, 4\ H/z)$	17	$(-1°, 2\ H/z)$
5	$(-3°, 0\ H/z)$	18	$(-1°, 3\ H/z)$
6	$(-3°, 1\ H/z)$	19	$(-1°, 4\ H/z)$
7	$(-3°, 2\ H/z)$	20	$(0°, 0\ H/z)$
8	$(-3°, 3\ H/z)$	21	$(0°, 1\ H/z)$
9	$(-3°, 4\ H/z)$	22	$(0°, 2\ H/z)$
10	$(-2°, 0\ H/z)$	23	$(0°, 3\ H/z)$
11	$(-2°, 1\ H/z)$	24	$(0°, 4\ H/z)$
12	$(-2°, 2\ H/z)$		

$F = \{0\ Hz, 1\ Hz, 2\ Hz, 3\ Hz, 4\ Hz\}$ and tail deflection angles $T = \{-4°, -3°, -2°, -1°, 0°\}$, see Table 1.

We assume an open space where the drone flies without obstacles, and we consider two scenarios: the *landing problem*, where the X-distance is within a range of 15 to 25 meters, and the *mid-range problem*, where the X-distance between starting and target points is within a range of 25 to 100 meters.

The optimization function is the total energy cost (battery) consumed by the ornithopter. Two main metrics will be used in the experiments: *precision* and *cost*. The precision measures how far the final state is from the target in the XZ-plane. The cost is given by the energy consumption determined by maneuvers performed by the ornithopter. The energy consumed by the ornithopter performing a certain maneuver for a time step t_s is given by the following formula:

$$E = t_s(K_{aero} f^3 + c_r). \tag{1}$$

The first term represents the energy consumption in the flapping wings maneuver. It has been empirically proven that this consumption is proportional to the cube of the flapping frequency [20]. The constant coefficient K_{aero} is estimated empirically based on the physical characteristics of the ornithopter[1]. The second term models the residual energy consumption c_r when the ornithopter is not flapping, mainly due to the onboard electronics. As the cost of moving the tail is negligible compared to the average consumption of the electronics, we consider the cost c_r constant[2]. As expected, Equation 1 shows that the energy cost is dominated by the flapping maneuvers.

[1]All the experiments in this paper used a value $K_{aero} = 2.5\ W/Hz^3$, obtained empirically for the ornithopter prototype used in [27].

[2]The empirically estimated upper bound of c_r is $5W$.

There are some approaches to address this optimization problem but unfortunately, solving the highly non-linear system that describes the dynamics of the ornithopter is time consuming and the methods from the literature are no valid to be used online. Thus, the main goal in this paper is to use supervised learning algorithms to compute short-term trajectory optimization for an ornithopter.

4 OTO Data Set

To our knowledge, the first data set available for ornithopter trajectory optimization was given by [25]. The data set was created to train NN's using the planner OSPA [27] with scenarios involving flapping, gliding and landing maneuvers. In this paper, we improve the data set provided in [25] by adding a great number of new flight trajectories. For the sake of reproducibility, we make our data set publicly available[3].

4.1 Data Generation

Our intention is to make a large and representative ornithopter flight data. To this end we create a set of initial states for the ornithopter describing many possible starting positions in terms of velocity and pitch. A simple translation is applied over the initial and target state to simulate that every trajectory starts at the origin of coordinates. Thus, we assume that the ornithopter always starts at $(0, 0)$ position in the XZ plane with null angular velocity. We pick 9 initial states by taking the combinations of pitch and speed values $P = \{-30°, 0°, 30°\}$ and $S = \{0.5\ m/s, 1\ m/s, 1.5\ m/s\}$, respectively.

From each of these initial states, we mimic the technique proposed in [27] to expand a tree of states by applying each maneuver in the set M consecutively taking a timestep of $0.1s$ and $1s$ between maneuvers for the landing and mid-range problems, respectively. We consider a sequence of maneuvers increasing by $1Hz$ in flapping frequency and $1°$ in tail deflection angle (see Table 1).

In order to create a large number of trajectories while avoiding exponential grow, we expand the tree of states and prune some vertices using a clustering approach at each level. This technique consists in grouping nearby states in space and keeping only the ones with lower cost in each cluster for the next iteration. With a big enough number of clusters by iteration, we get wide trees of low-cost trajectories scattered in space.

We randomly generate a large number of target states in the space within the spatial ranges shown in Table 2. Target states have fixed pitch and speed values of 30° and

[3]https://github.com/mpcutino/OTO_dataset/tree/master/data

Table 2 Ranges of target state position used in the experiments. MR: mid-range, L: landing

Problem	X range (m)	Z range (m)
L	[0, 25]	[0, −25]
MR	[25, 100]	[−50, 50]

0 m/s, respectively. This represents a suitable posture for the ornithopter to reach the target. For each target point and the expanded tree, we select all states within a ratio of 0.01 m and 1 m from the target for the landing and the mid-range problem, respectively. From the selected states, we take as optimal ending state the one with better pitch and speed compared to the target in the landing problem, and the one with lower energy consumption in the mid-range problem. We use these metrics because precision is more important than energy in the landing problem. For each of the optimal ending states we get its trajectory from the corresponding tree and include it into the data sets.

Following the above procedure we create over 27.026 different trajectories (a total of 345.676 states) for the landing problem, and 95.153 different trajectories (a total of 876.333 states) for the mid-range problem. From each trajectory, we store the state-control waypoints, the initial and target states, the timestep and the energy cost. Finally, we use the 80% of the data to train the models and the rest for testing.

4.2 Data Analysis

The ML-based algorithms proposed in this study use information about flight states and maneuvers to predict efficient trajectories between initial and target states. In this section we made a brief study over the maneuvers and over all variables in flight states in the OTO data set. With this study we pretend to observe the difficulty of our problem and to obtain some insights to improve the accuracy of predictions.

Recall that the tree of states in the OSPA algorithm is expanded using the set M of maneuvers in Table 1. Figure 2 shows the percent of use of each maneuver in the landing and middle range data sets. We can observe that all maneuvers are used in the mid-range data set, but there are some of them with very low use rate. In this cases, the difficulty of predicting the correct maneuvers increases. On the other hand, for the landing scenario, the percentage value of each maneuver used decreases. As a result, the maneuver corresponding to class 24 is not used by OSPA during the landing flights.

Figure 3 illustrates the data distribution of the landing dataset, using the difference between the target and current state of the ornithopter. The analysis using these values is

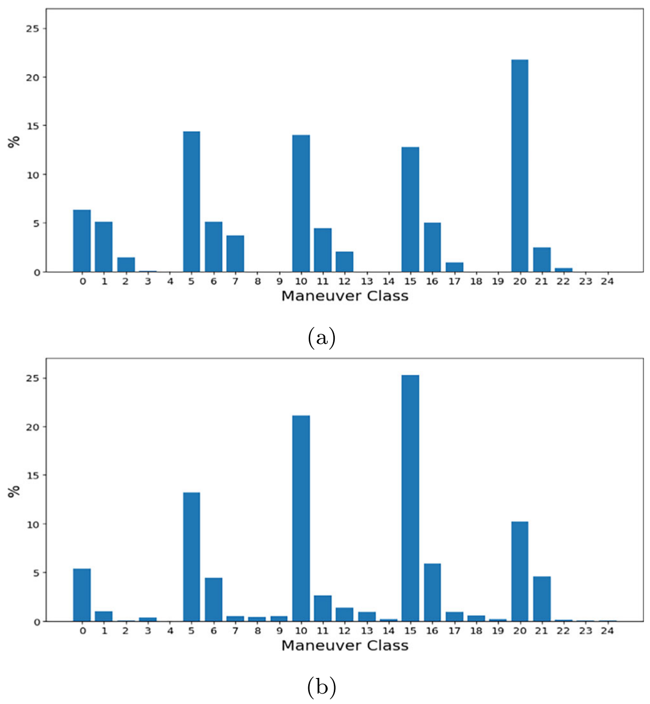


Fig. 2 (a) and (b) show the percent of use of the maneuvers in landing and mid-range problems, respectively

important as it highlights the distribution of each variable in the data set. In addition, the difference of the states is the input of our algorithms. As each variable represent a different distribution, in the training phase we scale its values to have zero mean and unit standard deviation. On the other hand, the box-plots across the action classes highlights the complexity of our problem. For each individual variable, there are several outliers, which increase the variability in the data, but decreases the statistical power.

We also performed a scaling operation on the training phase for the data corresponding to the mid-range problem. The distribution of the variables and the box-plots of the mid-range data set, result in a similar picture to the one generated for the landing scenario.

5 Machine Learning Algorithms

The OSPA planner performance depends on the maneuvers it takes at a given state. Our proposal relies on trajectories generated by OSPA, using different supervised ML algorithms. The goal is to obtain energy-efficient trajectories and reduce computational times compared to exploratory algorithms. Thus, computational cost is a key point in the design of the architectures of our models.

We propose two types of Artificial Neural Networks: a Multi-Layer Perceptron (MLP), to predict actions; and

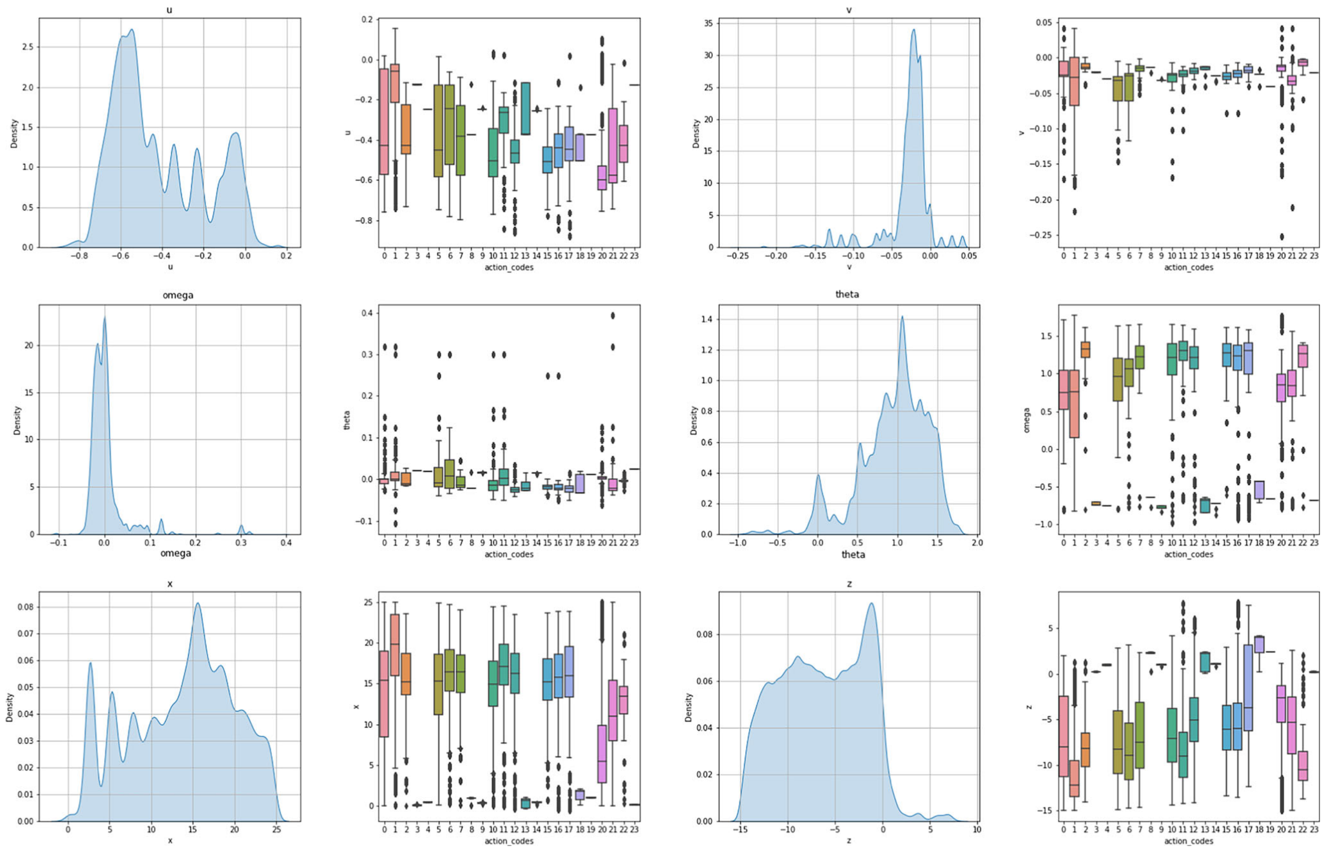


Fig. 3 Distribution of the landing problem dataset. The variables (u, v, omega, theta, x, z) represents the difference between the target and initial state. The distributions are depicted in the first and third plot with a Kernel Density Estimation (KDE) function. The box-plots in the second and fourth column shown the values of the variables across the different action codes, from 0 to 23

a Recurrent Neural Network (RNN), to predict states. For the MLP, we consider a discrete set of actions or maneuvers in a continuous space of configurations. This derives in two different kinds of problems, classification and regression, respectively. The advantage of the maneuver classification network is that it can generate feasible paths for the ornithopter according to the predefined dynamic model and, it is possible to compute the cost in terms of energy. However, this network only consider a discrete set of available maneuvers. Therefore, we propose a second type of network to directly predict a set of future states given the past states in the flight. In this case we tackle the continuous problem, even if the training data is generated in a discrete manner.

For the classification problem, we also propose the use of Random Forest (RF) [2], as an alternative to the MLP. Random Forest has proved to outperform classifiers based on neural networks in terms of recognition accuracy, stability, and robustness to features in several scenarios [6, 21, 36]. Moreover, RF produces fast responses when properly controlling the number of trees in the forest.

5.1 Multi-Layer Perceptron

MLP is a feed-forward artificial neural network that is usually trained in a supervised manner and contains several hidden layers. For our problem, we model a trajectory from the data as a discrete set of state-maneuver pairs. Since the set of maneuvers is fixed, we train the network to learn the next control maneuver depending on the current and target states.

The goal is to build a MLP to learn the best maneuver in each scenario. For this, we label the maneuvers with a class tag from $\mathcal{C} = \{0, 1, \cdots, N\}$ where N represents the maximum integer identifier of the proposed maneuvers, as in Table 1, depending on the evaluated problem. For the landing task, $N = 23$, and $N = 24$ for the mid-range flights. The input of the network is the difference between the current and target states, and the output is the maneuver to be performed.

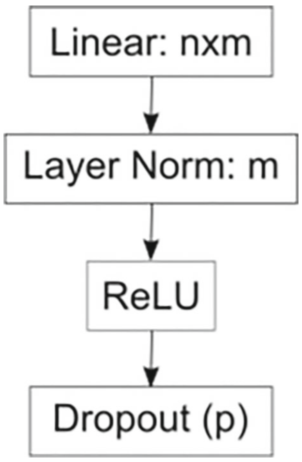


Fig. 4 Definition of a Basic Block for the MultiLayer Perceptron. It is composed by a Linear layer with input of size n and output of size m, a Normalization layer, a ReLU activation function, and a Dropout layer with probability p

5.1.1 MLP Architecture

The architecture of the network is designed as the application of consecutive Basic Blocks. The definition of a Basic Block is depicted on Fig. 4. First, a Linear layer transform a vector of size n into a vector of size m. The result is normalized applying the Layer Normalization technique described in [1]. We introduce non-linearities with the use of the ReLU activation function, and finally we add a Dropout layer to prevent complex co-adaptations between neurons [9].

The bells and whistles of the proposed MLP are depicted on Fig. 5. We use a Deep Neural Network with 8 modules, containing a total of 777.792 neurons. Residuals connections are applied between the second and the fourth Basic Block, and between the fourth and the sixth. This type of connection reduce the complexity of the learning task, making the network easier to optimize [8].

5.1.2 Training

The MLP was trained using the RMSprop optimizer with a loss function based on Cross-Entropy. The loss values are weighted to consider the different *a priori* probabilities of the classes adopted. The function is defined as:

$$L_{CE}(x) = -\sum_i w_i \, p_i(x) \log q_i(x), \tag{2}$$

where $p_i(x)$ is the probability of sample x of belonging to the class i, $q_i(x)$ is the probability prediction of the MLP, and w_i is a weight function to take into account the different probabilities of belonging to class i. The weight value per class is computed as:

$$w_i = (1 - P_i)^{\alpha_i}, \tag{3}$$

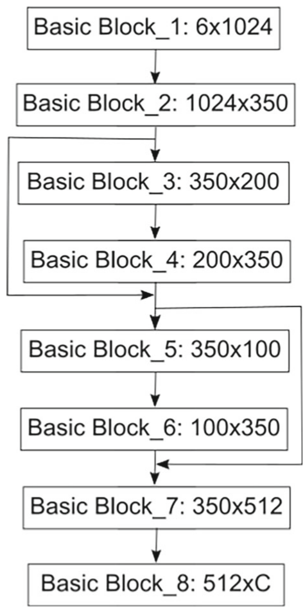


Fig. 5 Architecture of the proposed MLP. The arrows between non-consecutive blocks represents residual connections. The parameter C in the output layer is the number of classes depending on the problem: $C = 24$ for landing, and $C = 25$ for mid-range flights

where P_i is the *a priori* probability of class i and it is estimated as the percentage of samples of class i in the training dataset, and α_i is a parameter that highlights the class relevance. w_i is useful to compensate the differences in the representation of each class and, at the same time, it allows to *push* the network prediction in a specific direction to reduce (increase) the false negatives (positives).

5.1.3 Path Generation

Once trained, the MLP can predict the control maneuver m from a flight state s. Also, we implement the kinodynamic model introduced in [27], $K(s, m)$, to compute the next state given an initial state and a control maneuver. This allows us to compute the full trajectory by using the next formula in a loop starting at the initial state:

$$s_{i+1} = K(s_i, \mathcal{N}(s_i)),$$

where $\mathcal{N}(s_i)$ represents the action predicted by the network for the state s_i. The loop is stopped when s_{i+1} is further than s_i from the target state.

5.2 Recurrent Neural Network (RNN)

The goal of the proposed MLP is to predict the next maneuver at each time step using the information of the current state. Note that this strategy is dependent from the ornithopter model. We propose a RNN to predict the next flight state based on previous states. Recurrent architectures have been widely adopted for solving forecasting problems, with the use of Gated Recurrent Units (GRU) [28, 29] or Long Short-Term Memory (LSTM) neurons to handle sequential relationships [33].

In this case, we use a decoder architecture so only the initial distance is needed to develop a complete trajectory from it. As can be seen in the Fig. 6, a decoder has a single input, and produces a sequence starting from this input. For our specific case, the input is the initial distance, and the output is a set containing the next states in the trajectory.

5.2.1 RNN Architecture

We build a simple RNN with one recurrent layer and one output layer with a linear activation function. The output layer has as many units as flight states considered for prediction (two states for the mid-range problem and six states for the landing problem). The input is the initial distance from the current to the target state, and the output is the set of distances between each element in the trajectory and the target state. The RNN is executed recursively to get the complete trajectory starting from the initial distance. All input variables are normalized following the same strategy as in the MLP scenario. This represents an efficient strategy to push the network to predict low values while keeping the ability to compute the next state. The next state can be obtained by simply subtracting the output of the network and the target state. The recurrent layer has 11 LSTM neurons. We select LSTM neurons to deal with the vanishing gradient problem encountered by traditional recurrent neurons. In addition, we decide to keep a small architecture to obtain a fast predictions rate.

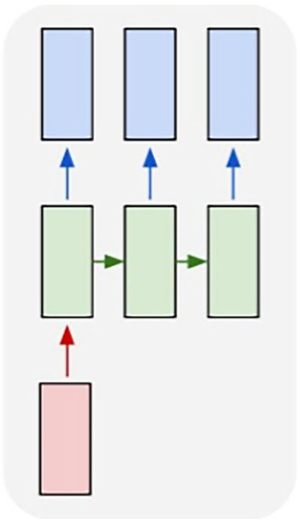


Fig. 6 Overview of the decoder RNN architecture

5.2.2 Training

The network is trained with one trajectory at a time, by comparing its output with the expected values. The goal of the training is to obtain the same input sequence shifted by one step ahead. The optimization process is carried out by the Adam optimizer using the values recommended in [14]: a learning rate of 0.001, a β_1 value of 0.9, a β_2 value of 0.999, an epsilon value of 1e-8 and no weight decay. The learning is stopped after 100 epochs. The loss function used is the MSE (Mean Squared Error):

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(d(y_i, \hat{y}_i)\right)^2,$$

where N is the number of considered flight states, y_i is the i component of the OSPA state and $\hat{y}_i$ is the i-component of the RNN predicted state. In our specific case, minimizing the MSE implies that the likelihood function between the predicted states distribution and the OSPA states is maximized.

5.2.3 Path Computation

At every time step t, the information to predict the next state consists on the previous prediction of the distance (y_{t-1}) at time step $t-1$, and the recurrent layer hidden states (h_{t-1}) at the same time step. The hidden states condenses all previous information regarding the trajectory; see Fig. 7 for a representation of the proposed RNN, unrolled until time step $t-1$. The algorithm ends when an output y_t is close enough to zero, i.e. the target has been reached and the trajectory is over. For $t = 0$, we use as input the difference between the target and the initial state.

5.3 Random Forest

The combination of different models over the same classification problem is usually known as ensemble learning.

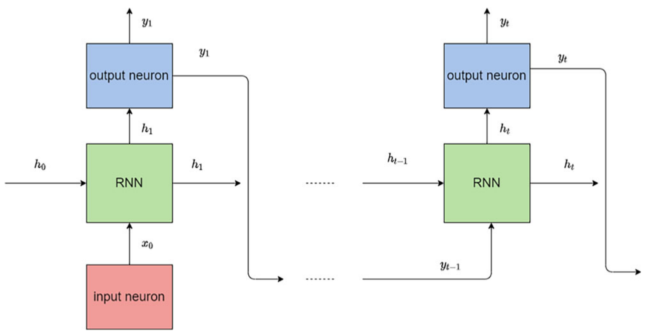


Fig. 7 Decoder RNN neuron. The input neuron (red box) contains six units, one for each variable of the ornithopter state

In the case of RF, the predictions are obtained by a voting process over the output of different Decision Tress [26]. To produce different distributions in the models, every tree selects randomly a subset of the features present in the data, and random samples from the dataset.

To evaluate the importance of the attributes before creating a new node in the trees, we selected the Gini Index (GI) as it results in higher precision on the conducted experiments. GI is calculated by subtracting the sum of the squared probabilities of each class from one.

$$GI = 1 - \sum_{i=1}^{C} p_i^2, \tag{4}$$

where C represent the number of classes and p_i is the probability of class i, obtained as the frequency of appearance on the considered split.

The RF architecture for training is obtained by applying a Grid Search strategy. For each task, i.e. landing and mid-range flights, we define a set of possible parameters, and we create different forest using all possible combinations. Each forest is evaluated on a small portion of the training data, i.e. a validation set, and the best configuration of parameters is maintained for full training. The specific values of the parameters are depicted on Section 6. In addition, the path generation follows the same guidelines detailed in the MLP section.

6 Experiments

In this section we compare the proposed ML methods against the OSPA planner[4]. The metrics used are the average values of energy consumption (*Cost*), execution time (*Time*), the Euclidean distance between the last state and the target (*Precision*) and the mean distance between the calculated and OSPA trajectories (*Error*). The precision is calculated by using the x and z values of the states. The Error is defined as the average distance between a fixed number of points in trajectories generated by OSPA and the ML algorithms. The points are obtained by sampling 10 random states for each trajectory and computing the correspondent positional distance. This metric gives us an idea about the similarity between two given trajectories. Metrics are computed over the testing dataset.

[4]It is important to notice that our problem is very specific from an engineering point of view, and we need to integrate complex and non-linear dynamics. This makes it hard to find heuristics that works properly. This is why we decided to only compare with OSPA, which is particularly suited for the problem at hand.

The classifiers are compared using the metrics Accuracy, average Recall, and Macro average Recall. In a binary setting, Recall and Accuracy are defined as:

$$Recall = \frac{TP}{TP + FN}, \tag{5}$$

$$Accuracy = \frac{TP + TN}{N}, \tag{6}$$

where TP, FN, TN represent True Positive, False Negative and True Negatives samples, respectively. N is the cardinality of the test set. As our problem is not binary, we compute the Accuracy as the fraction of correctly classified samples. On the other hand, Recall is computed per class, thus transforming our problem in C binary tasks. The average Recall is computed as the weighted average of the recall per class. The weight is defined by the number of samples of the class in the test set. In addition, the Macro average metric does not take into account the number of samples; therefore, it assumes equal importance on every class. This value is a representation of the behaviour on classes less populated on the dataset. To counteract the class imbalance, the α parameter selected in the loss function of the MLP is 5.

In both scenarios, mid-range and landing, the 20% of the training dataset is used for validation. Validation data is used differently depending on the algorithms. For the neural networks, we keep the model with lower loss value on the validation data; for Random Forest, we perform a gird search to optimize the model parameters.

The ML algorithms were tested on a Intel Core i5-8250U CPU, with a clock frequency of 1.60GHz. The NNs were optimized using the nvidia graphic card GeForce MX150. We use Pytorch [24] for the implementation of the MLP, and Keras for implementing the RNN.

6.1 Landing Problem

Landing phase remains to be one of the most crucial and difficult tasks to achieve among the flight envelope of an aerial robot. Landing requires complex maneuvers in order to prepare the ornithopter to perform a perching operation. Moreover, there is a need for a fast computation because the ornithopter has limited time for reaction. Also, for the landing problem it is more relevant to end the trajectory closer to the target rather than reducing the energy cost.

In the learning phase for this task, RNN and MLP are trained by a maximum of 100 and 300 epochs, respectively. The validation dataset is used to keep the model with the best loss value across all the epochs. In this manner, we ensure a better generalization when predicting over unseen data. For the RF algorithm, we perform a grid search over the validation data, resulting in a model that uses 100 trees, the Gini Index as the quality function, and a maximum depth

Table 3 Accuracy and Recall of the proposed classifiers on the test data set for the landing problem. M-AVG Recall stands for Macro AVG Recall

Algorithm	Accuracy	AVG Recall	M-AVG Recall
ANN [25]	80.13	80	49
MLP	91.59	92	67
RF	**99.82**	**99**	**98**

of 25 for every tree. The classifiers uses 24 classes in this scenario. For its part, the RNN considers six states to train and predict.

6.1.1 Results

A comparison between the classifiers is depicted in Table 3. Both MLP and RF perform with high accuracy and recall, but RF obtains the best results by a large margin. Moreover, the Macro average metric reflects a drop in the performance of MLP for the underrepresented classes on the test set, while RF maintains its excellent results. The performance of the Artificial Neural Network (ANN) proposed in our previous work [25] is also evaluated. The goal is to evidence the impact of the improvements conducted in this work: better performance for more complex algorithms using a dataset 10 times larger.

Table 4 shows the performance of the algorithms compared to OSPA. We added two new metrics, (*V-error*) and (*P-error*), measuring the error in the last state in terms of speed (m/s) and pitch (rad), respectively. Since the trajectory in the landing scenario must end just before a perching maneuver, the goal now is to predict trajectories very close to the target with suitable velocity and pitch angle, specifically 0 and 30°, respectively.

The experiments highlight the capabilities of the ML-based methods to significantly improves computational time without a meaningful loss in the other metrics. It is worth noting that although the velocity error is not negligible, this result is similar to the one obtained by the OSPA algorithm. On the other hand, RF outperforms both the networks and the OSPA on almost all metrics. This result is an expression of the learning capabilities of the ensemble of trees. The algorithm has learned an efficient optimization function, that is capable to outperform the heuristic used to collect the training data. Moreover, this increased performance is obtained on previously unseen samples. To push further the study of performance under different conditions, we design two additional experiments.

Table 4 Results for the landing problem

Algorithm	Cost	Time (s)	Precision (m)	Error	V-error	P-error
OSPA	21.61	37.24	0.07	na	0.63	0.81
MLP	22.30	0.21	0.17	0.54	0.63	0.77
RNN	na	0.08	0.74	1.01	0.63	0.45
RF	20.72	2.3e-3	0.06	0.01	0.63	0.80

6.1.2 Robustness

The ornithopter is equipped with sensors to know its situation and the relative location of the target. These sensors are subject to noise; therefore the actual position and state are known with some degree of incertitude and noise.

We simulate the uncertainty by adding a random Gaussian noise to the state vector at each time step, defined as:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5(\frac{x-\mu}{\sigma})^2}. \quad (7)$$

The states are normalized prior to the noise injection. We consider a zero mean ($\mu = 0$) distribution, but we vary the value of the standard deviation σ, to compare the performance of the algorithms under different conditions. Table 5 shows the effect of the perturbation. It can be noticed how the performance degrades as the noise increase. We

Table 5 Results for the landing problem assuming a Gaussian noise on every state of the trajectory. MLP shows robustness to noise even on hard conditions, i.e. when $\sigma = 1$, which can be equivalent to moving the ornithopter more than 10 meters from its original position

Sigma	Algorithm	Cost	Precision	V-error	P-error
0.01	MLP	22.39	**0.18**	0.63	0.76
	RNN	na	0.73	**0.46**	**0.62**
	RF	**21.09**	0.45	0.64	0.83
0.02	MLP	21.82	**0.20**	0.63	0.77
	RNN	na	0.77	**0.45**	**0.62**
	RF	**21.35**	0.73	0.65	0.85
0.05	MLP	22.49	**0.35**	0.63	0.75
	RNN	na	0.83	**0.45**	**0.68**
	RF	**20.82**	1.14	0.66	0.88
0.1	MLP	22.64	**0.59**	0.63	0.71
	RNN	na	1.19	**0.44**	**0.67**
	RF	**20.53**	1.52	0.66	0.86
0.2	MLP	22.63	**0.92**	0.62	**0.62**
	RNN	na	1.42	**0.45**	0.77
	RF	**21.29**	2.26	0.68	0.82
0.5	MLP	23.04	**1.72**	0.63	**0.51**
	RNN	na	2.88	**0.45**	0.86
	RF	**21.93**	3.01	0.70	0.75
1	MLP	22.77	**2.93**	0.65	**0.51**
	RNN	na	9.02	**0.39**	0.65
	RF	**21.10**	4.16	0.73	0.74

can conclude that the drone is able to reach the neighborhood of the target despite applying noise to the states. Moreover, a few remarks can be done:

- MLP outperforms the other algorithms in terms of precision by a large margin.
- RNN has a lower precision than RF on small noise, i.e. for $\sigma \in \{0.01, 0.02\}$. However, its performance is boosted for harder noise values, except $\sigma = 1$. In addition, RNN obtains the lower velocity error for all values of σ.
- RF and MLP performance is acceptable even on hard noise, i.e. when $\sigma = 1$. In terms of position, this value represents changes up to $13.7m$ in the X axis and $6.4m$ in the Z axis.
- The V-error and P-error are not highly affected under different configurations. This is due to the fact that the OSPA planner does not have constraints on the velocities and pitch values, thus they are not optimized. In addition, a high standard deviation affects these variables differently, as their values fall in a smaller range when compared to the positional variables.

Table 6 Results for the replanning problem when sigma varies

Sigma	Algorithm	Cost	Precision	V-error	P-error
0.01	MLP	22.37	0.25	0.63	0.78
	RNN	na	0.74	**0.46**	**0.61**
	RF	**20.22**	**0.14**	0.63	0.79
0.02	MLP	22.32	0.35	0.63	0.79
	RNN	na	0.70	**0.46**	**0.62**
	RF	**20.22**	**0.28**	0.63	0.79
0.05	MLP	22.54	0.69	0.63	0.79
	RNN	na	**0.67**	**0.45**	**0.60**
	RF	**20.25**	0.68	0.63	0.80
0.1	MLP	22.43	1.31	0.64	0.78
	RNN	na	**0.85**	**0.45**	**0.62**
	RF	**20.04**	1.29	0.64	0.80
0.2	MLP	22.63	2.77	0.62	**0.62**
	RNN	na	**0.84**	**0.44**	0.66
	RF	**20.14**	2.56	0.63	0.81
0.5	MLP	22.29	6.09	0.60	**0.78**
	RNN	na	**1.06**	**0.45**	**0.78**
	RF	**20.31**	5.68	0.63	0.82

6.1.3 Replanning

In real scenarios, the flight environment can be constantly changing. Therefore, replanning ability is critical for adapting to unforeseen events. For instance, the target changes and the agent must re-plan the path.

In order to simulate a re-planning in the middle of a trajectory, we randomly change the target position when the trajectory reaches the fifth waypoint. This new position is given by adding to the normalized target a Gaussian offset of zero mean and standard deviation σ (see Equation 7). As the value of σ can drastically change the target value on the X coordinate, we discard the noise if the new x value of the target position is lower than the x value of the current state. We also consider a lower upper bound for σ compared to the previous scenario. The upper bound for σ translates in a maximum change of $6.8m$ for the x value and $3.2m$ for the z value.

Table 6 shows the precision reached by the algorithms with regard to sigma. Note that the error is no longer relevant, as we have no equivalent OSPA trajectory to compare with. On this new scenario, both RF and MLP obtains a similar performance in terms of precision, being RF a little bit more accurate when replanning. It has to be noticed how the performance drop faster if compared to the previous experiments. This behaviour can be explained by the nature of the experiments. Changing the target position can result on a configuration that is not reachable from the current state. However, in the previous experiment, the state of the ornithopter is changed on every point of the trajectory; therefore, if the current configuration is not reachable, there is a chance that the next random noise results in a better setting.

Another relevant insight obtained from Table 6 is that RNN is capable of better adaptation when applying medium and hard noise ($\sigma > 0.05$) to the target state. This behaviour is obtained because the proposed RNN is independent from the ornithopter model. Therefore, in principle, any target state is reachable. In addition, as in the previous experiment, RNN excels on the velocity and pitch error when compared to the classifiers.

Finally, note that the RNN estimates the optimal trajectory as a sequence of states, but the required actions are not provided, thus a trajectory controller is further needed. In order to mitigate the effects of disturbances and unmodeled dynamics, a robust trajectory tracking controller could be used; see [30, 35] as some examples.

6.2 Mid-Range Problem

Performing long flights is one of the main advantages of ornithopters, as they optimize energy consumption by switching from flapping maneuvers to gliding. In this scenario, we consider the mid-range problem as the task of reaching a given configuration in a location that is far from the current position of the drone. The target setting has no restrictions on the variables involved in the state of the ornithopter.

For this new scenario, we apply the same methodology as for the landing problem, in terms of data generation, data preprocessing, network arquitecture and path generation. For RF, the best parameters obtained when performing the grid search over the validation data are: a forest with 200 trees, the Gini Index as quality function, and no restrictions over the maximum depth of the trees. The number of considered maneuvers in this approach is 25.

6.2.1 Results

Table 7 shows the performance of the proposed classifiers for the mid-range problem. We add the results computed with the neural network classifier used in [25], evidencing once again the improvements obtained by expanding the dataset, and by adding more complexity to the neural network structure and loss function. In addition, RF outperforms MLP by a large margin in all the metrics evaluated. However, the average Recall and the Macro average recall is similar in both methods. This translates in a similar performance across different maneuvers, despite the class imbalance present on the data. The high values obtained in theses metrics accentuate the importance of the α parameter used in the loss function during training, for the MLP. On the other hand, RF is more naturally robust against the imbalance as every tree learns from a random subset of the data. One aspect to notice is that the Macro average has a higher value for the MLP in this task when compared to the value obtained in the landing problem. This can be explained by the number of samples in the underrepresented classes. In the landing task, the least populated class has 14 items, while in the mid-range problem, it has 144.

Table 8 displays a comparison between OSPA, MLP, RNN, and RF for the metrics related to flight performance. Note that RNN does not compute maneuvers, therefore the value of Cost can not be calculated. However, we assume that a predicted trajectory with a small value of Error has a similar cost to the OSPA solution. The experiments show that ML algorithms drastically outperform the OSPA algorithm in execution time. In addition, MLP and RF are capable of lower energy consumption. However, the saving in battery results in a lower precision; i.e. the ML algorithms end further from the target state. In terms of error, the trajectories generated by MLP and RNN has an average distance to OSPA of 4 and $6m$, respectively. Compared to the landing case, the learning process struggles to adapt to the OSPA trajectories. This is highly related to the integration time. In the landing scenario, each waypoint on the trajectory is computed every $0.25s$ of flight

Table 7 Accuracy and Recall of the proposed classifiers on the test dataset for the mid-range flights. RF outperforms MLP and ANN by a large marging in all the metrics computed. M-AVG Recall stands for Macro AVG Recall

Algorithm	Accuracy	AVG Recall	M-AVG Recall
ANN [25]	69.2	69	63
MLP	80.0	80	81
RF	**97.7**	**98**	**98**

Table 8 Comparison results (average) for the mid-range problem. na is the acronym used for not applicable values

Algorithm	Cost	Time (s)	Precision (m)	Error
OSPA	74.11	524	0.68	na
MLP	65.34	0.22	1.26	4.33
RNN	na	0.07	2.71	6.42
RF	69.74	2.7e-3	1.44	0.72

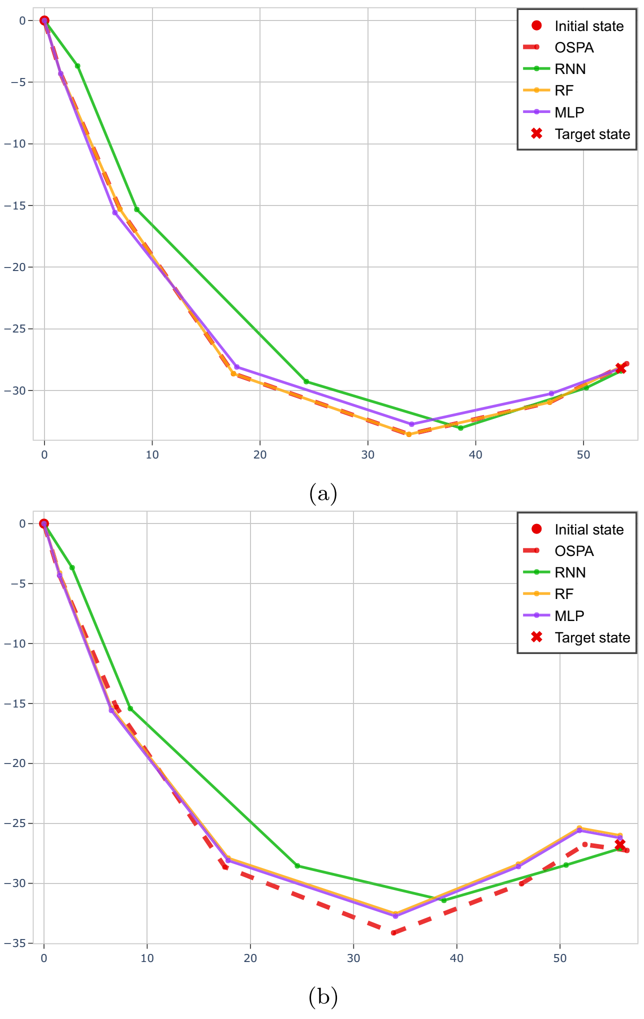
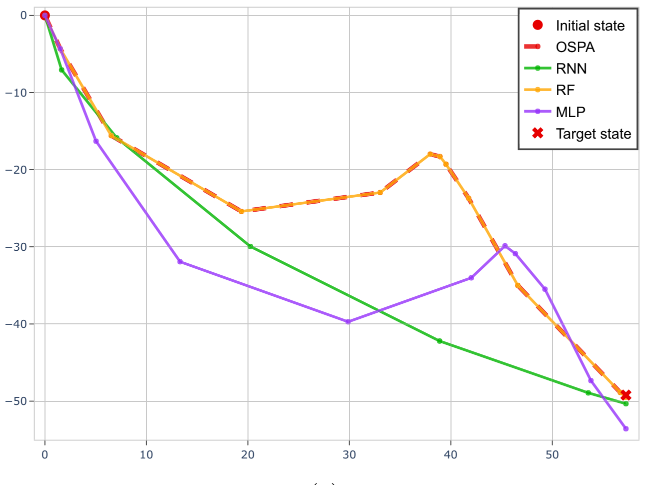
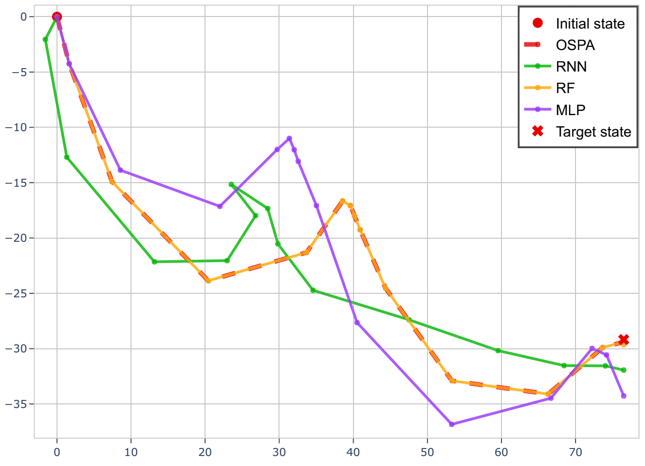


Fig. 8 Trajectories for OSPA, RNN, RF and MLP in the XZ-plane for the mid-range problem. When a given trajectory perfectly match other trajectory, then it is occluded on the drawings. The order of drawings is defined on the legend of the figures. Trajectories obtained with OSPA are drawn with dashed lines



(a)



(b)

Fig. 9 Examples of bad performance of MLP and RNN in the mid-range testing dataset. The same drawing principle of Fig. 8 was followed

time, while in the mid-range problem, this value increases to $1s$. A higher integration step produces higher errors when actions are predicted incorrectly. Moreover, it provides less opportunities to the algorithms to adapt. However, as in the landing scenario, RF produces the most accurate results. The trajectories are similar to the ones produced by OSPA, which is a direct consequence of the high accuracy values obtained in the testing data set.

Figure 8 shows good examples of the path generated using the ML algorithms, in terms of x and z components. In general, RF produces trajectories with a high similarity to those obtained with OSPA. In addition, it can be noticed the capabilities for reaching a given target even when the predictions does not match the states obtained with OSPA, as for RNN in Fig. 8 (a), and the three ML algorithms in Fig. 8 (b).

We found some situations where the MLP and RNN does not perform as well as expected. Figure 9 illustrates examples of bad convergence. For instance, the RNN has not been able to capture some trajectory behaviours, such as a sudden ornithopter climb before continuing its descent, as in Fig. 9 (a). This limitation is due to the simplicity of the RNN architecture, as it only contains 11 hidden neurons, which are insufficient for this type of trajectory complexity.

Another limitation of the proposed RNN is depicted on Fig. 9 (b). This algorithm is independent of the kynodynamic model of the ornithopter. Therefore, some trajectories can not be followed by the UAV. However, a simple *not-moving-back* behaviour can be implemented by only connecting points in the trajectory that moves forward. In addition, in most of the settings, RNN behaves well and tends to get near the target. On the other hand, for MLP is sometimes harder to obtain a similar trajectory to the one produced by OSPA, specially when the initial prediction is wrong, as in both examples of Fig. 9.

We believe that there is a strong correlation between the predictions of the ML algorithms and the target elevation in z compared to its position in x. To support this intuition, we carried out a study depending on the ratio $\mid z/x \mid$. As a convention, we term as long trajectories to those where $\mid z/x \mid \leq 0.4$; short trajectories to those where $\mid z/x \mid > 0.8$; and balanced trajectories in any other case. We acknowledge that $\mid z/x \mid \leq 0.4$ does not necessarily means that the corresponding trajectory is long. However, this is the case in our data since this result is obtained when the x value is high. A similar analysis can be established for the rest of the intervals defined.

Table 9 shows different values for the metrics according to the positional ratio. Each metric is obtained from trajectories that lies on a spatial rectangle defined by the ratio.

Table 9 Performance depending on the rectangle ratio. The best result per algorithm and metric is highlighted

$\mid z/x \mid$	Cost	Precision	Error	Total paths
	MLP | RNN | RF	MLP | RNN | RF	MLP | RNN | RF	
(0.8, 1]	59.59 | na | 42.30	1.23 | **0.57** | 0.59	**2.08** | 4.40 | 0.30	2772
(0.6, 0.8]	**40.44** | na | **40.65**	1.40 | 0.60 | 0.57	3.98 | 2.75 | 0.50	4129
(0.4, 0.6]	43.86 | na | 46.12	**0.89** | 2.21 | 0.55	4.64 | 6.4 | 0.80	8537
(0.2, 0.4]	100.9 | na | 115.0	1.60 | 4.05 | 3.10	4.95 | 5.8 | 0.91	8247
[0, 0.2]	147.0 | na | 302.4	0.94 | 1.92 | **0.50**	3.48 | **0.72** | **0.16**	104

The higher the ratio, the more similar are the absolute values of z and x. This study helps the user to make the decision on which algorithm to use depending on the scenario. It must be noticed the reduction of $Cost$ for the trajectories found in the interval of (0.4, 1]. This is explained by the maneuvers performed on each case. For long distances, i.e. values in the interval [0, 0.4], some flapping maneuvers has to be performed to reach the target. On the other hand, short and balanced intervals provide a better environment for gliding, which translates in a lower cost.

Another important aspect to notice from Table 9 is the behaviour of RF. In all scenarios, the ensemble of trees is capable of providing trajectories similar to OSPA, meaning a high precision when reaching the target. This value is only affected for the interval (0.2, 0.4], where MLP is by a large margin, the most precise algorithm. In the case of RNN, a better precision is obtained for high ratios. This may be because for longer trajectories flapping is needed, and RNN misses long-term patterns.

7 Conclusions

In this paper, a novel data set for ornithopter trajectory optimization, OTO, is constructed. To our knowledge, OTO data set is the first data set with a high number of pseudo-optimal trajectories for ornithopter motion planning. Two different scenarios are defined: mid-range flights, and the landing task. For each scenario, a train-test split is made publicly available to provide a fair point of comparison among different algorithms. In addition, we release the code related to the ornithopter model. This model can be used in future researches for increasing the data, or testing other techniques, such as Reinforcement Learning.

We provide a benchmark on our data set using different algorithms targeting two tasks: classification of maneuvers, and direct prediction of the next state. As classifiers, we propose a novel MLP architecture with residual connections, and a Random Forest. Both classifiers obtains similar metrics to those provided by OSPA and, in addition, they can be used online. We emphasize the use of RF, which is an understudied technique in the aerial robotic community when compared to other ML-based algorithms. Our proposed RF algorithm is capable to outperform OSPA, even in the average precision measure for the landing scenario; i.e. the supervised algorithm overcomes the planner used to collect the data. We also propose a RNN as the regression approach. Using this strategy, it is possible to obtain trajectories without using the mathematical model of the ornithopter. This algorithm is specially efficient for re-plannig, i.e. when the target state is changed during the flight. Moreover, our algorithms, in particular MLP, are robust under the injected noise which simulates wrong sensors readings.

Future lines of research includes the investigation of better architectures for the Recurrent Neural Network. Furthermore, regression techniques for predicting the tail deflection and the flapping frequency can be implemented to compare with the approach exposed in this work. In addition, other machine learning techniques can be explored, both for comparison and for extending the use cases of our proposal. For instance, algorithms of Reinforcement Learning can be designed to study the optimal trajectory in the presence of obstacles. Finally, performing experiments with a real ornithopter can provide a feedback for combining our planners with nonlinear controllers for flight stabilization and trajectory tracking.

Acknowledgements This work is partially supported by the Spanish Ministry of Economy and Competitiveness (MTM2016-76272-R AEI/FEDER,UE), the Spanish Ministry of Science and Innovation CIN/AEI/10.13039/501100011033/ (PID2020-114154RB-I00) and European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement #734922.

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by M.A. Pérez-Cutiño, F. Rodríguez and L.D. Pascual. The first draft of the manuscript was written by J.M. Díaz-Bañez and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript. Conceptualization and supervision was mainly performed by J.M. Díaz-Bañez.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Code or data availability The Ornithopter Trajectory Optimization (OTO) data set and evaluation code can be found at https://github.com/mpcutino/OTO_dataset.

Declarations

Ethics approval All of the authors confirm that there is no potential acts of misconduct in this work, and approve of the journal upholding the integrity of the scientific record.

Consent to participate The authors consent to participate.

Consent for Publication The authors consent to publish.

Conflict of Interests The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv:1607.06450 (2016)
2. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)
3. Chai, R., Savvaris, A., Tsourdos, A., Chai, S.: Overview of trajectory optimization techniques. In: Design of Trajectory Optimization Approach for Space Maneuver Vehicle Skip Entry Problems. Springer, pp. 7–25 (2020)
4. Coutinho, W.P., Battarra, M., Fliege, J.: The unmanned aerial vehicle routing and trajectory optimisation problem, a taxonomic review. Comput. Indust. Eng. **120**, 116–128 (2018)
5. DeLaurier, J.D.: An ornithopter wing design. Canadian aeronautics and space journal **40**(1), 10–18 (1994)
6. Han, T., Jiang, D., Zhao, Q., Wang, L., Yin, K.: Comparison of random forest, artificial neural networks and support vector machine for intelligent diagnosis of rotating machinery. Trans. Inst. Meas. Control. **40**(8), 2681–2693 (2018)
7. Hausknecht, M., Stone, P.: Deep recurrent q-learning for partially observable Mdps. In: 2015 Aaai Fall Symposium Series (2015)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016)
9. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580 (2012)
10. Horn, J.F., Schmidt, E.M., Geiger, B.R., DeAngelo, M.P.: Neural network-based trajectory optimization for unmanned aerial vehicles. J. Guidance Control Dynam. **35**(2), 548–562 (2012)
11. Ilin, R., Kozma, R., Werbos, P.J.: Beyond feedforward models trained by backpropagation: a practical training tool for a more efficient universal approximator. IEEE Trans. Neural Netw. **19**(6), 929–937 (2008)
12. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: Stomp: Stochastic trajectory optimization for motion planning. In: 2011 IEEE International Conference on Robotics and Automation. IEEE, pp. 4569–4574 (2011)
13. Kelly, M.: An introduction to trajectory optimization: How to do your own direct collocation. SIAM Rev. **59**(4), 849–904 (2017)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 (2014)
15. Kosari, A., Maghsoudi, H., Lavaei, A., Ahmadi, R.: Optimal online trajectory generation for a flying robot for terrain following purposes using neural network. Proceedings of the Institution of Mechanical Engineers Part G: Journal of Aerospace Engineering **229**(6), 1124–1141 (2015)
16. Lu, Y., Yi, S., Liu, Y., Ji, Y.: A novel path planning method for biomimetic robot based on deep learning. Assembly Automation (2016)
17. Mirzaei, M., Kosari, A., Maghsoudi, H.: Optimal path planning for two Uavs in a pursuit-evasion game. In: 2021 IEEE International Conference on Automation/XXIV Congress of the Chilean Association of Automatic Control (ICA-ACCA). IEEE, pp. 1–7 (2021)
18. Mordatch, I., Todorov, E.: Combining the benefits of function approximation and trajectory optimization. In: Robotics: Science and Systems, vol. 4 (2014)
19. Mordatch, I., Todorov, E., Popović, Z.: Discovery of complex behaviors through contact-invariant optimization. ACM Transactions on Graphics (TOG) **31**(4), 1–8 (2012)
20. Nguyen, T.A., Phan, H.V., Au, T.K.L., Park, H.C.: Experimental study on thrust and power of flapping-wing system based on rack-pinion mechanism. Bioinspiration & Biomimetics **11**(4), 046001 (2016). https://doi.org/10.1088/1748-3190/11/4/046001
21. de Oliveira, G.G., Ruiz, L.F.C., Guasselli, L.A., Haetinger, C.: Random forest and artificial neural networks in landslide susceptibility modeling: a case study of the fão river basin, southern brazil. Nat. Hazards **99**(2), 1049–1073 (2019)
22. Otte, M., Correll, N.: C-forest: Parallel shortest path planning with superlinear speedup. IEEE Trans. Robot. **29**(3), 798–806 (2013)
23. Park, J.H., Yoon, K.J.: Designing a biomimetic ornithopter capable of sustained and controlled flight. Journal of Bionic Engineering **5**(1), 39–47 (2008)
24. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al: Pytorch: an imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems **32**, 8026–8037 (2019)
25. Pérez-Cutiño, M., Rodríguez, F., Pascual, L., Díaz-Báñez, J.: Neural Networks Algorithms for Ornithopter Trajectory Optimization. In: 2021 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, pp. 1665–1670 (2021)
26. Quinlan, J.R.: Induction of decision trees. Machine Learning **1**(1), 81–106 (1986)
27. Rodríguez, F., Díaz-Báñez, J.M., Sanchez-Laulhe, E., Capitán, J., Ollero, A.: Kinodynamic planning for an energy-efficient autonomous ornithopter. Computers & Industrial Engineering **163**, 107814 (2022)
28. Salloom, T., Kaynak, O., He, W.: A novel deep neural network architecture for real-time water demand forecasting. J. Hydrol. **599**, 126353 (2021)
29. Salloom, T., Kaynak, O., Yu, X., He, W.: Proportional integral derivative booster for neural networks-based time-series prediction: Case of water demand prediction. Eng. Appl. Artif. Intel. **108**, 104570 (2022)
30. Salloom, T., Yu, X., He, W., Kaynak, O.: Adaptive neural network control of underwater robotic manipulators tuned by a genetic algorithm. J. Intell. Robot. Syst. **97**(3), 657–672 (2020)
31. Suarez, A., Perez, M., Heredia, G., Ollero, A.: Small-Scale Compliant Dual Arm with Tail for Winged Aerial Robots. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 208–214 (2019)
32. Woo, M.H., Lee, S.H., Cha, H.M.: A study on the optimal route design considering time of mobile robot using recurrent neural network and reinforcement learning. J. Mech. Sci. Technol. **32**(10), 4933–4939 (2018)
33. Wu, H., Yan, W., Xu, Z., Li, S., Cheng, T., Zhou, X.: Multimodal prediction-based robot abnormal movement identification under variable time-length experiences. Journal of Intelligent & Robotic Systems **104**(1), 1–15 (2022)
34. Yijing, Z., Zheng, Z., Xiaoyi, Z., Yang, L.: Q Learning Algorithm Based Uav Path Learning and Obstacle Avoidence Approach. In: 2017 36Th Chinese Control Conference (CCC). IEEE, pp. 3397–3402 (2017)
35. Yu, X., He, W., Li, H., Sun, J.: Adaptive Fuzzy Full-State and Output-Feedback Control for Uncertain Robots with Output Constraint. IEEE Transactions on Systems Man, and Cybernetics Systems (2020)
36. Zekić-Sušac, M., Has, A., Knežević, M.: Predicting energy cost of public buildings by artificial neural networks, cart, and random forest. Neurocomputing **439**, 223–233 (2021)

37. Zhang, B., Liu, W., Mao, Z., Liu, J., Shen, L.: Cooperative and geometric learning algorithm (cgla) for path planning of uavs with limited information. Automatica **50**(3), 809–820 (2014)
38. Zhang, B., Mao, Z., Liu, W., Liu, J.: Geometric reinforcement learning for path planning of uavs. Journal of Intelligent & Robotic Systems **77**(2), 391–409 (2015)

Miguel Angel Pérez-Cutiño is a PhD student in Mathematics at the University of Seville, Spain. He has received a bachelor's degree in Computer Science from University of Havana in 2018, and a master's degree in Logic, Computing and Artificial Intelligence from University of Seville in 2021. He is currently working at VirtualMech developing computer vision systems for fault detection. His research interests include computational geometry, computer vision and deep learning.

Fabio Rodríguez is in his second year of the Ph.D. program in Mathematics at the University of Seville. His work is focus on the areas of algorithms, computational theory, and mathematics and recently, data science and machine learning. He is passionate about the algorithms design and analysis and currently, he is applying computational techniques to the resolution of problems with drones in industry. He has participated in many international conferences related to algorithms and machine learning. Fabio's research is supported by the DENiM European Grant Agreement (https://cordis.europa.eu/project/id/958339). He holds a Master degree in Mathematicsin University of Seville and a Bachelor of Computer Science from the University of Havana. See more info at https://www.linkedin.com/in/fabio25-rodriguez.

Luis David Pascual Callejo is an aeronautical engineer at Airbus, working currently as product owner of an Airbus developed web application. During his professional career he has gained experience in aircraft production, web development and project management. He has a double Master of Sc.in aeronautical engineering between the University of Madrid and Supareo (Toulouse), an MBA at College des Ingenieurs (Paris) and a master's in mathematics at the University of Seville. His research interests are mainly focused on machine learning applied to aeronautics. See full bio at https://www.dpascual.space.

José-Miguel Díaz-Báñez is a Full Professor of Applied Mathematics at the University of Seville, Spain. His research interests include computational geometry, decentralized algorithms for cooperative UAVs and computational ethnomusicology, within the common theme of geometric algorithms. He has coauthored 75 research papers in indexed journals by the Journal Citation Reports, 3 books, 15 book chapters and more than 80 conference papers. He has organized several international conferences related to geometric algorithms. Prof. Díaz-Báñez has supervised 8 PhD's thesis and 12 Master's thesis. He has led 15 research projects, most of them funded by the European Commission and Spanish Government. He is currently the coordinator of the Optidrone project, as well as other projects on geometric algorithms applied to path planning with drones. See more info at http://alojamientos.us.es/galgo/ and https://diazbanez.wordpress.com/.

Affiliations

M. A. Pérez-Cutiño[1,2] · **F. Rodríguez**[1] · **L. D. Pascual**[3] · **J. M. Díaz-Báñez**[1]

F. Rodríguez
frodriguex@us.es

L. D. Pascual
d.pascualcallejo@gmail.com

J. M. Díaz-Báñez
dbanez@us.es

1 Department of Applied Mathematics II, University of Seville, Seville, Spain
2 VirtualMech, Seville, Spain
3 Airbus, Seville, Spain