Automating Fixed Wing Forced Landings with Offline Reinforcement Learning

*Alexander Quessy, Thomas Richardson, and Sebastian East
University of Bristol, Queens Building, United Kingdom, BS8 1TR

ABSTRACT

Executing off-field landings in unprepared locations is a crucial skill for single-engine piston fixed-wing aircrew, particularly during sudden engine failures. The unpredictability and challenging nature of such failures often leads to flight crew overload and accidents. Further, as autonomous air vehicle capabilities advance, incorporating the ability to autonomously land following engine failure is likely to become a vital design requirement.

This paper presents a unified forced landing methodology that leverages reinforcement learning (RL) to develop adaptable policies for a wide range of forced landing scenarios, with a focus on model-based offline RL. The research outlines a graphical simulation environment, procedures, and algorithms for training an RL-based controller. The effectiveness of the proposed approach is demonstrated, highlighting that offline RL is a promising solution for designing controllers capable of executing glide approaches into predetermined locations.

1 INTRODUCTION

Executing off-field landings in unprepared locations is a key skill required by all Single Engine Piston (SEP) Fixed Wing (FW) aircrew as engine failures or fires can occur suddenly and unexpectedly at any point in flight. Executing a dead-stick/no-power landing is challenging and requires the aircrew to fly the aircraft within an appropriate airspeed range to avoid stalling or over-speeding. They must also navigate to a safe landing spot, all while not being able to maintain a fixed altitude for a pre-planned approach. The suddenness of engine failures can complicate matters further, as the aircrew must quickly identify a landing site to navigate to before losing too much altitude and the option to land. The suitability of a landing site may only become apparent when the aircraft is low and near the final landing location. At this point, there may not be enough energy available to land elsewhere.

Regulations reflect this limitation requiring operators to glide-clear [1], remaining 1,000 feet above built-up-areas and preventing scheduled commercial air transport on SEP FW aircraft, within Europe. Despite forced landing training being an integral part of SEP aircrew training and assessment, incidents involving engine failures still tragically result in fatal accidents due to a loss of control and failure to identify suitable landing sites [2, 3, 4].

The numerous concurrent tasks that must be completed to safely execute a forced landing place large cognitive demands upon the aircrew, these include, but are not limited to:

- Identifying that a loss of power has occurred and taking appropriate immediate actions to prevent loss of control.
- Control of the aircraft through the use of visual reference and instruments, which may become inoperative if reliant on the engine driven vacuum pump.
- Visual identification of an appropriate landing location, which often depends on the aircraft's attitude, requiring the aircrew to maneuver the aircraft to visualize the locations depending upon the airframe geometry.
- Navigating towards the designated landing zone whilst accounting for wind direction and speed.
- Attempting engine restart, shutdown or fire control/suppression procedures as appropriate.
- Coordinating with air traffic to arrange search and rescue operations after the forced landing.

As a result, a common cause of engine failure incidents leading to accidents is aircrew overload, where startle following an engine failure leads to either a loss-of-control or a failure to position the aircraft to make an approach towards an appropriate landing location [5]. Further, as the size and capabilities of autonomous pilot-less aircraft gradually increase, SEP FW aircraft are a sensible platform owing to their ubiquity and low operational cost (compared to multi-engine and turbine types). It is reasonable to expect that autonomous off-field engine-out landings will become a certification requirement to enable these types of operations.

Designing methods to automatically solve this task is challenging as it requires a balance between online exploration and exploitation, while acting upon limited information with high-stakes outcomes. Reinforcement learning (RL) offers a solution to this problem by enabling policies to be learned that can generalize to complex tasks through the use of previous experiences.

*Email address: aq15777@bristol.ac.uk

This work presents a unified forced landing methodology that aims to serve two purposes: assisting aircrew in executing forced landings to help reduce startle, surprise and overload and establishing the groundwork for autonomous fixed-wing forced landings. Our contributions can be summarized as follows:

- Section 3 presents the general challenge of landing a fixed-wing aircraft in an off-field location, and describes a 2D graphical simulation environment to simulate aircraft landing following an engine failure in various locations. Source code for this environment at the following URL . . ..
- Section 4 details the procedures and algorithms used to generate offline data for training an RL-based controller, their relevance to the fixed-wing landing problem, and their application to model-based offline RL.
- Section 5 showcases the effectiveness of offline RL in designing controllers capable of executing glide approaches into predetermined locations by learning from demonstrations. The performance of the learned controller is evaluated in a variety of both in-distribution and out-of-distribution scenarios.

2 Related Work

2.1 Fixed-Wing Forced Landings

Previous research on fixed-wing off-field landings is fairly limited. [6], focused on identifying landing sites through edge detection algorithms applied to real-world data. The authors of [6] used 3D Dubin's path-planning methods [7] to generate trajectories, whereas our work employs sampling-based approaches, constraining the aircraft to a fixed rate of descent and airspeed. Unlike our approach, [6] allows the aircraft's pitch angle, and therefore rate of descent, to vary. Additionally, [8] and [9] considered landing site reachability analysis.

2.2 Model Based Offline Reinforcement Learning

RL is a type of machine learning that uses a reward signal from an agent to train a policy capable of achieving task specific goals [10]. Combining deep neural network based function approximators with RL has proven to be hugely successful to solve a variety of complex tasks including games [11], robotics [12] and natural language processing [13]. This work focuses on the offline RL setting, which involves learning from a dataset of offline demonstrations. Behavior Cloning (BC) is the simplest version of this, where the policy is directly learned from (s, a) state-action pairs, under the assumption that the demonstrations come from an optimal policy demonstrator π^* [14].

Unlike model-free RL methods such as SAC [15] and DDPG [16], model-based offline RL methods, such as MoREL [17] and MOPO [18], learn both a model and a policy, enabling online planning similar to Model Predictive Control (MPC) [19]. This approach results in an improved online policy compared to pure BC examples, thanks to the added benefits of planning. This work uses Model-Based Offline Planning (MBOP) [20] to learn a value function that enables zero-shot online adaption to new goals or constraints. MBOP also provides improved interpretability by allowing the loss and accuracy of the dynamics and value function to be viewed directly throughout the training process.

3 Problem Statement

This section provides a detailed description of the motivation and methodology behind the FW forced landing simulation environment. It also explains how this problem can be formulated as a Markov Decision Process (MDP) and how RL can be used to solve it. The source code for the environment is available at the following URL [released with full paper] for use by other researchers.

3.1 Fixed-Wing Environment

The `flyer` simulation environment models aircraft dynamics using a 3D Dubin's car, represented by the dynamic equations specified in equation 1. The aircraft's position is represented by a 3D point in space, denoted as (x, y, z), with a heading angle specified as θ. The aircraft's velocity, V, rate of descent $\dot{Z}$, and the simulation time step δ_t are pre-defined and fixed. Wind and atmospheric effects are not incorporated in the simulation, hence all headings are considered as tracks and airspeeds as ground speeds.

Only bank angle ϕ is used to control the aircraft's position and heading, constrained to a maximum of 30 degrees. The aircraft's turn-rate is then calculated using c_ϕ which causes the aircraft to turn at a standard rate of 3 $^\circ/s$ at 15 $^\circ$ angle of bank. Bank angle is then used as the action in the MDP formulation.

$$\begin{aligned} x_{t+1} &= x_t + V cos(\theta)\delta t \\ y_{t+1} &= y_t + V sin(\theta)\delta t \\ z_{t+1} &= z_t - \dot{Z}\delta t \\ \theta_{t+1} &= \theta_t + c_\phi \phi \delta t \end{aligned} \tag{1}$$

Observations consist of either an 800x800 RGB image captured from a downward-facing camera with a 30-degree field of view, gimbled to be level along the xy earth plane from the aircraft, as shown in Figure 1. Additionally, the aircraft's state is represented as a tuple (x, y, z, θ). For this report, only the aircraft's position and attitude observation is considered, but the RGB image is left within the simulation and provided source code for future work.

The termination criterion is met when the aircraft comes to a stop on the ground, which can be either due to a crashed state or when the ground roll is completed. A crashed state occurs when the aircraft's preassigned *health* property falls to zero, by default this is an integer set to 1. Colliding with an obstacle like a rock usually causes damage reducing health

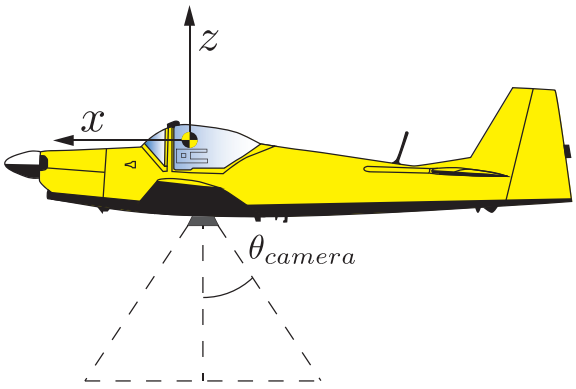


Figure 1: Image observations are generated by a simulated camera, $\theta_{camera} = 30°$.

by one. Some obstacles like crops are destroyed in this event, others like trees persist inflicting damage at the next time step.

The simulated world comprises several landing terrain types each with varying levels of complexity, from easiest to hardest:

1. *Grass*: an open grass field, the entire area is unobstructed and safe for landing.
2. *Orchard*: a grass field containing sparse uniformly distributed apple trees, collision with a tree causes the aircraft to crash. The field also contains stumps and rocks that cause one damage as the aircraft passes over the object.
3. *Crops*: a dirt field containing crops, collision with a crop deals one damage and removes the crop. Crops are clustered together as a 2D Gaussian distribution.
4. *Forest*: a dirt field containing trees, collision with a tree causes the aircraft to crash. Trees are clustered together as a 2D Gaussian distribution.
5. *Beach*: a narrow sand area containing banana trees. The area is thin and only exists where water meets land.
6. *Water*: a water area, any landing in this region results in a crash.

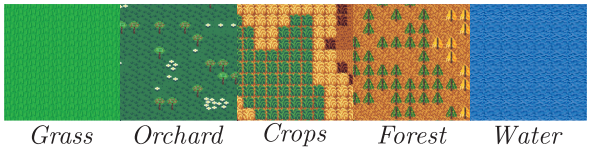


Figure 2: Landing terrain types in the flyer environment, with order of landing difficulty/danger increasing from left to right

The Flyer environment is procedurally generated offering the user large amounts of control as to how the environment is

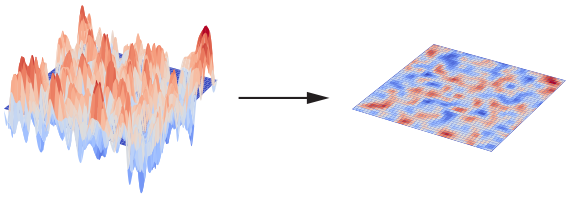


Figure 3: OpenSimplex (Perlin) noise used to generate land and water tile types on the map

constructed, whilst also allowing for a huge range of environmental diversity. 3D OpenSimplex Noise, similar to Perlin noise [21], is initially used to smoothly separate the environment into land and sea and beach terrain types with beach being placed along the intersection. The noise function returns a single value for each (x, y, z) point in 3D space, sweeping across all $(x, y, z = 0)$ positions and returning the noise value at each. If the noise value is less than the water noise threshold the tile is assigned as water, otherwise it is a land type of tile, this is shown in Figure 2.

To generate fields $n_{partition}$ points are randomly placed across the ground xy plane and a nearest neighbor Voronoi decomposition [22] is used to create regions. Each region is assigned a terrain type using a multi-noulli distribution based upon the prior probability of each terrain type existing within the environment, a $(256, 256)$ texture map is created as a result. Objects, such as trees and bushes, are placed within terrain types using the probability distribution for each object within the terrain type.

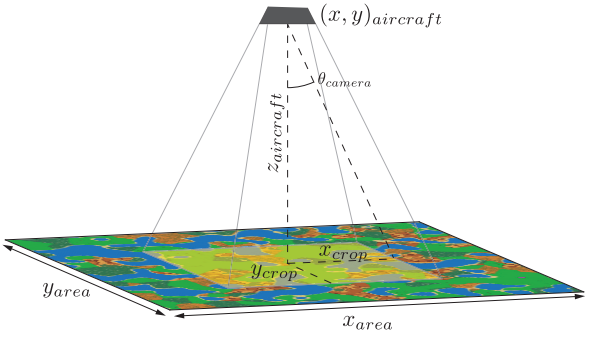


Figure 4: Crop to render aircraft image observation

A base image is rendered by tiling 16×16 RGB textures onto the $(256, 256)$ texture map, objects are placed on-top of this image by selectively replacing pixels in appropriate locations. The resulting 4096×4096 base image is cropped based on the aircraft camera's location. This image size is set using $z \tan(\theta_{camera})$ and the crop centered around the aircraft's location, as shown in Figure 4. The 30° camera angle used for the aircraft crop simulation is equivalent to a

35mm full frame camera lens.

3.2 MDP Formulation

Based on our simulation we can define the problem of fixed wing landings as an MDP, defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$. The goal is to learn a policy $\pi(\mathbf{a}|\mathbf{s})$ that maximizes the expected cumulative discounted reward of the MDP.

- $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively.
- $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ maps states and actions to a probability distribution over subsequent states.
- $R(\mathbf{s}, \mathbf{a})$ represents the reward function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.
- $\gamma \in (0, 1]$ is the discount factor.
- $\pi_\beta(\mathbf{a}|\mathbf{s})$ is the behavior policy used to generate the dataset $\mathcal{D} \sim P_{\pi_\beta}$, which is itself used to train the online policy $\pi_\phi(\mathbf{a}|\mathbf{s})$.
- $s_0 \sim \mu$ is the starting distribution of the MDP.

The objective of RL is to find the policy π^* that maximizes the expected discounted sum of rewards:

$$\pi^*(s_t) = \arg\max_{a \in \mathcal{A}} \left\{ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi^*(s_t)) \right\}$$

In the `Flyer` MDP reward is sparse and binary:

$$R(s) = \begin{cases} 1, \text{if safe-landing} \\ 0, \text{if crashed} \end{cases}$$

Assuming that the real environment is not available during training, we have access only to a dataset of demonstrations $\mathcal{D}$ containing tuples of (s_t, a_t, r_t, s_{t+1}). The objective in offline learning is to learn a policy that can achieve zero-shot online task adaption by using the offline dataset. It is also assumed that the policy used to generate these demonstrations, while effective in reaching the goal, is suboptimal.

4 METHODOLOGY

This section outlines the methods used to locate a suitable landing site and train an RL-based controller to identify and execute an approach into the destination. The process is viewed as a 2-step approach that involves generating an offline dataset from simulation to train the controller, followed by training an offline RL controller to execute the approach into the destination.

To generate a dataset of sub-optimal landing trajectories, we use an Importance Sampling (IS), Monte-Carlo Tree Search (MCTS) based algorithm where the value of each node is found using the L_2 distance from a target position and optimal trajectory, found by backtracking along nodes connected by edges with the maximal value. The target position is found by searching for the Pole of Inaccessibility (PoI) [23] on all grass arrays using the grass terrain map.

4.1 Target Position Search

Algorithm 1 provides a description of the procedure used to calculate the PoI. We assume access to the full terrain map creating a binary array, where positions with grass fields are assigned 1 and everything else is 0. A point is assigned to be the PoI if the sub-array formed by expanding linearly around the point being tested of distance $dist$ is all grass with value 1, and the sub-array does not exceed the bounds of the problem space. We show this search procedure graphically in figure 5.

Algorithm 1 PoI Search

Require: T_{grass} grass terrain map
$dist_{max} \leftarrow 0$
for (i_x, i_y) in T **do**
  $dist = dist_{max} + 1$
  while BOUNDSCHECK(i_x, i_y, t, $dist$) **do**
    if $T_{grass}[i_x : i_x + dist, i_y : i_y + dist] == 1$ **then**
      $dist_{max} = dist$
      $\mathbf{x}_{\text{PoI}} = [i_x, i_y, 0]$
    end if
  end while
end for
return $\mathbf{x}_{\text{PoI}}$

Algorithm 2 Within Bounds Check

procedure BOUNDSCHECK(i_x, i_y, $array$, $dist$)
  if $i_x - dist < 0$ **then**
    return False
  end if
  if $i_y - dist < 0$ **then**
    return False
  end if
  if $i_x + dist > array.x_{length}$ **then**
    return False
  end if
  if $i_y + dist > array.y_{length}$ **then**
    return False
  end if
end procedure

4.2 Monte-Carlo Tree-Search

An IS-MCTS algorithm [24] is used to generate a trajectory from the initial starting position $\mathbf{x}start$ to the final goal position $\mathbf{x}goal$. The action sampling in this algorithm is weighted based on the distance from the goal state. Figure 6 illustrates a simplified example of the IS-MCTS algorithm. After initialization IS-MCTS is a 3-step process of *selecting* which $n_{samples}$ edges to expand along the horizon length H_n. The flyer environment is used to *simulate* the roll-out of each selected node by H_n steps. The tree is then *expanded* by adding the new edges and vertices into the tree and the

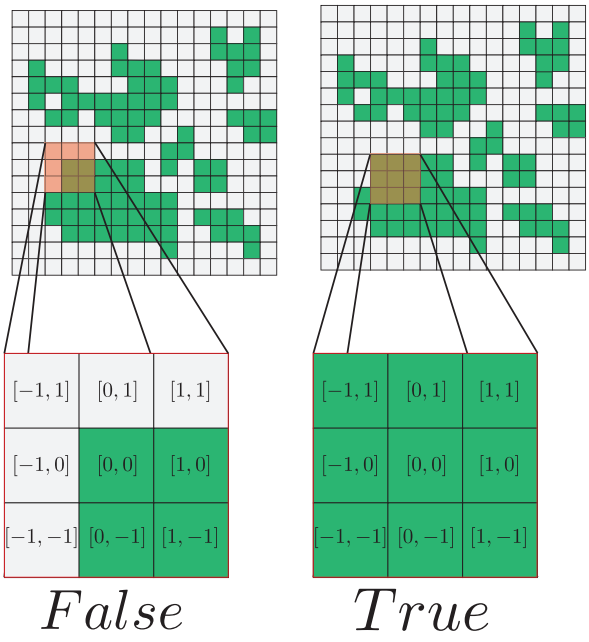


Figure 5: Target Point Grass Check

probability of each node is updated to reflect this weighting. Once the maximum number of steps has been sampled, back-propagation is performed along nodes with the highest value to calculate the optimal trajectory.

Algorithm 3 provides the full IS-MCTS algorithm used in the flyer environment where $\mathbf{x}_{goal} = \mathbf{x}_{\text{PoI}}$, found using algorithm 1. The following functions are used through IS-MCTS to obtain useful demonstrations:

- f_{dyn}: Aircraft dynamics from the flyer environment, as described in equation 1.
- f_u: Control sampling function, modelled as a Gaussian distribution where the mean is the last bank angle and variance is fixed at 0.2, $u_t \sim \mathcal{N}(\mu = u_{t-1}, \sigma = 0.2)$.
- f_{constr}: Space constraint function that returns $true$ if (x, y, z) position values are in the range $[[0 < x < 255], [0 < y < 255], [0 < z < 64]]$ and $false$ otherwise.
- f_x: A multinomial distribution, where the probability mass function can be formulated using the gamma function as:

$$\frac{\Gamma \sum_i x_i + 1}{\Pi_i \Gamma(x_i + 1)} \Pi_{i=1}^{k} p_i^{x_i}$$

The vertex in V with the maximum value is closest to the goal state $\mathbf{x}goal$. From there, the edges contained in G are backtracked to select the optimal trajectory τ_{opt}. This is illustrated graphically in Figure 7, sampled with hyperparamaters: $N_{steps} = 2000$, $N_{samples} = 100$, $H_n = 10$.

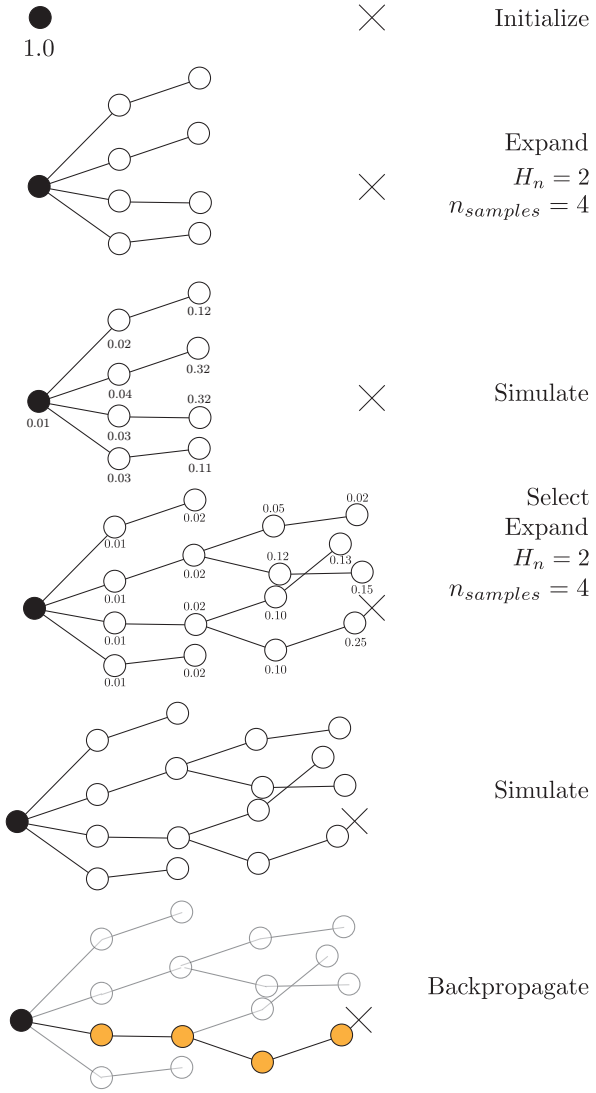


Figure 6: Simplified 2D IS-MCTS algorithm

4.3 *Model Based Offline Planning*

Using the data generated by the previously described goal searching and navigation algorithms, MBOP (Model Based Offline Planning) a model-based offline RL algorithm that combines a BC policy with an iterative guided n-step shooting method. Classical supervised learning is used to train the model on the dataset $\mathcal{D}$ containing $n_{episodes}$, this includes 3 components:

1. A one-step dynamics model, $f_{dyn} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathbb{R}$, such that $(r_t, s_t) = f_{dyn}(s_t, a_t)$.
2. A behavior cloned policy, $\pi : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, such that $a_t = \pi(s_t, a_{t-1})$.
3. A truncated value function, $V : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, providing the expected return over horizon N of taking action a in state s, such that $\hat{R}_H = V(s_t, a_{t-1})$.

Algorithm 3 IS-MCTS

Require: $\mathbf{x}_{start}$, $\mathbf{x}_{goal}$, H_n, $N_{samples}$, N_{steps}, f_{dyn}, $f_u(\mathbf{x}) = \Pr(\mathbf{U} = \mathbf{u}|\mathbf{X} = \mathbf{x})$, f_{constr}
  $G = (\mathbf{x}_{start}, \emptyset)$ ▷ Create list of edges and vertices
  $V = \{\mathbf{x}_{start} : \|\mathbf{x}_{goal} - \mathbf{x}_{start}\|_2\}$ ▷ Create dictionary of vertex values
  $p_v \leftarrow \{\mathbf{x}_{start} : \frac{V[\mathbf{x}_{start}]}{\sum V}\}$ ▷ Probability of selecting an edge v at point $\mathbf{x}$
  for N_{steps} **do**
    for i in N_{values} **do**
      $p_v[\mathbf{x}_i] = \frac{V[\mathbf{x}_i]}{\sum V}$ ▷ Update probability
    end for
    $f_{\mathbf{x}} = \Pr(\mathbf{x}_0, \ldots, \mathbf{x}_k; p_v[\mathbf{x}_0], \ldots, p_v[\mathbf{x}_k])$
    for i in $N_{samples}$ **do**
      $\mathbf{x}_i = f_{\mathbf{x}}()$ ▷ Sample from multinomial distribution
      for i_h in H_n **do**
        $\mathbf{u}_{i+i_h} \leftarrow f_{sample}(\mathbf{x}_{i+i_h}, \mathbf{u}_{i+i_h})$
        $\mathbf{x}_{i+i_h+1} \leftarrow f_{dyn}(\mathbf{u}_{i+i_h}|\mathbf{x}_{i+i_h})$
        if $f_{constr}(\mathbf{x}_{i+i_h+1}) == False$ **then**
          $G \cup (\mathbf{x}_{i+i_h}, \{\mathbf{x}_{i+i_h} : \mathbf{x}_{i+i_h+1}\})$
          $V \cup \{\mathbf{x}_{i+i_h+1} : \|\mathbf{x}_{goal} - \mathbf{x}_{i+i_h+1}\|_2\}$
        end if
      end for
    end for
  end for

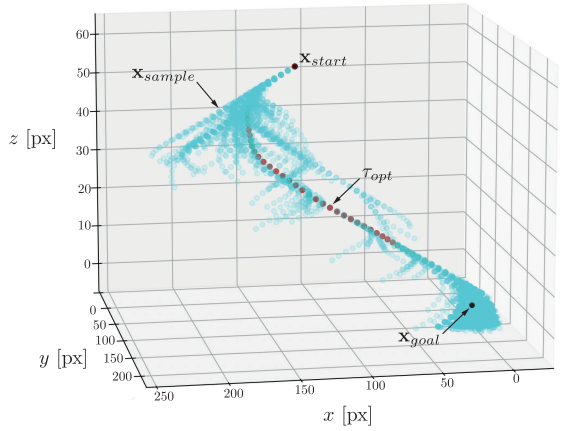


Figure 7: MCTS Results: sampled points are shown in blue and the final trajectory points in red. Vertices not shown for clarity.

Each component is represented with a deep neural network, with input and output layers of the same size as the function shown above, and 2 hidden layers of size 500. After experimentation, we found that ensemble network representations were not required in comparison to the original MBOP algorithm [20]. The samples from MCTS are obtained from a uniformly distributed cube with a start position $\mathbf{x}start = [x, y, z]$, where $x \sim \mathcal{U}[x0 - 10, x_0 + 10]$, $y \sim \mathcal{U}[y_0 - 10, y_0 + 10]$, $z \sim \mathcal{U}[z_0, z_0 - 10]$, and $\mathbf{x}0 = [128, 128, 64]$, which is the center-point of the map. The target position for all examples is $\mathbf{x}goal = [14, 198, 0]$. Only the MCTS samples that successfully land within the target field are used to train MBOP. The 3 components described above are trained over 1×10^6 steps using 500 episodes, as shown in Figure 8.

Online MBOP-Trajopt is used to sample trajectories over a horizon length H_n using a combination of sample based Model Predictive Control (MPC) and the behavior cloned policy π. Mixing of the noisy behavior cloned action $a_t = \pi(s_t, a_{t-1}) + \epsilon$, and the best last action $\tau_{a_{i=min(t,H-1)}}$ is controlled with a mixing parameter β. Algorithm 4 provides a full description of the online MBOP trajectory optimization procedure.

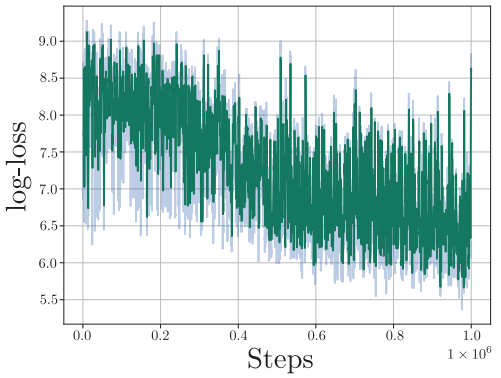
Algorithm 4 MBOP-TrajOpt

Require: Dynamics Model f_d, Policy π, Value V, Horizon H_n, MDP Dynamics P, MDP termination T
  $\mathbf{s} \sim \mu$ ▷ Initial state
  $\tau_a = [0_0, \ldots, 0_{H_n}]$
  $terminated = False$
  while $terminated == False$ **do**
    $\mathbf{R}_N = 0_N$ ▷ N trajectory returns
    $\mathbf{A}_{N,H} = 0_{N,H}$ ▷ N action trajectories of length H
    for $n = 1 \ldots N$ **do**
      $s_1 = \mathbf{s}, a_0 = \tau_a, R = 0$
      for $t = 1 \ldots H_n$ **do**
        $\epsilon \sim \mathcal{N}(0, \sigma^2)$
        $a_t = \pi(s_t, a_{t-1}) + \epsilon$ ▷ BC Action Sampling
        $\mathbf{A}_{n,t} = (1 - \beta)a_t + \beta\tau_{a_{i=min(t,H-1)}}$
        $(s_{t+1}, r_{t+1}) = f_{dyn}(s_t, \mathbf{A}_{n,t})$
        $R = R + r_{t+1}$
      end for
      $\mathbf{R}_n = R + V(s_{H_n+1}, \mathbf{A}_{n,H})$
    end for
    $\tau_a = \frac{\sum_{n=1}^{N} e^{\kappa \mathbf{R}_n} \mathbf{A}_{n,t+1}}{\sum_{n=1}^{N} e^{\kappa \mathbf{R}_n}}$
    $\mathbf{s} = P(\mathbf{s}, \tau_{a_0})$
    $terminated = T(\mathbf{s})$
  end while

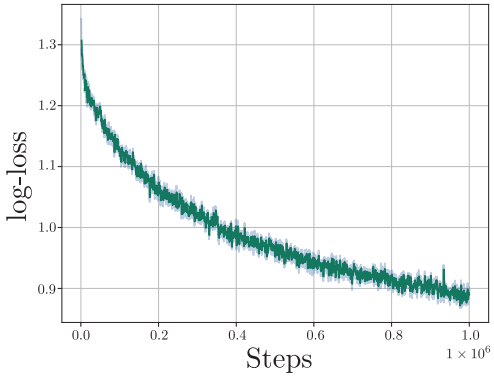
5 EXPERIMENTS

Results comparing MCTS to pure BC and MBOP with TrajOpt are presented in Figure 9. There is clear improvement in both the reliability and accuracy of the task with MBOP and BC compared to MCTS. It was found that a 100% task success rate could not be achieved using BC or MBOP due to constraint violations resulting from crashing into obstacles or leaving the boundaries of the flyer environment, as defined in f_{constr}.

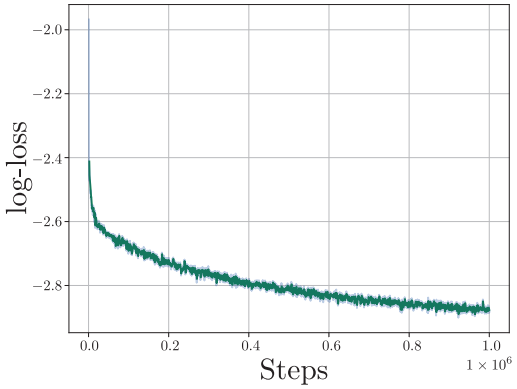
1. Operating too far from the known dynamics so plans



(a) Value V Loss



(b) Policy π Loss



(c) Dynamics f_{dyn} Loss

Figure 8: Loss Plots for each of MBOP's components, 100 data point rolling average is shown in green and the underlying data in blue.

are not accurate.

2. We violated the constraints of the flyer environment (as described in $f_{constraint}$).

NOTE TO REVIEWER: Further results will be presented in final paper including training over a larger number of environment seeds, along with a comparison of the effect of dataset quantity on MBOPs accuracy and final task performance. We will also include the full hyper-parameters in a table within the appendix. We will also include the URL link to a public github repository

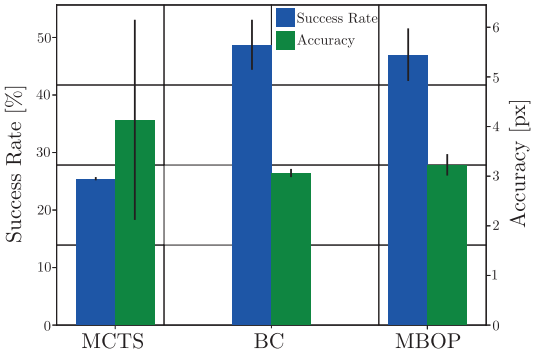


Figure 9: Comparison of methods over 5 seeds, using 100 samples, 1SD is shown with error bars.

6 CONCLUSIONS AND FUTURE WORK

This work presents a methodology for autonomous landing control based entirely on offline data. Although the datasets used were obtained from computer simulations, future research would benefit from using real-world offline data collected from an actual aircraft. The study demonstrates how the planning function enables detection of possible constraint violations by the aircraft, making our approach useful as an assistive tool for flight crew.

REFERENCES

[1] J Hanafin. (uk) standardised european rules of the air, exceptions to the minimum height requirements. *ORS4*, 01, 2021.

[2] UK Air Accidents Investigation Branch. Aaib investigation to grumman aa-5, g-bbsa. *Air Accidents Investigation Branch*, 2022. Loss of power after takeoff, Teesside International Airport, 25 September 2021.

[3] UK Air Accidents Investigation Branch. Aaib investigation to piper pa-46-350p (modified), g-hyza. *Air Accidents Investigation Branch*, 2022. Propulsion system failure and forced landing, one mile north of Cranfield Airport, 29 April 2021.

[4] NTSB. Report era14la383, n50xv. *National Transportation Safety Board*, 2014.

[5] Javier Rivera, Andrew B. Talone, Claas Tido Boesser, Florian Jentsch, and Michelle Yeh. Startle and surprise on the flight deck: Similarities, differences, and prevalence. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 58(1):1047–1051, 2014.

[6] Luis Mejias, Daniel Fitzgerald, Pillar Eng, and Xi Liu. Forced landing technologies for unmanned aerial vehicles : towards safer operations. 2009.

[7] Giuseppe Ambrosino, Marco Ariola, Umberto Ciniglio, Federico Corraro, Alfredo Pironti, and Micuel A. De Virgilio. Algorithms for 3d uav path generation and tracking. *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 5275–5280, 2006.

[8] Matthew Coombes, Wen-Hua Chen, and Peter Render. Reachability analysis of landing sites for forced landing of a uas in wind using trochoidal turn paths. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 62–71, 2015.

[9] Matthew Coombes, Wen-Hua Chen, and Peter Render. Landing site reachability in a forced landing of unmanned aircraft in wind. 10 2016.

[10] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[11] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[12] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision, 2022.

[13] OpenAI. Gpt-4 technical report, 2023.

[14] Dean A. Pomerleau. *ALVINN: An Autonomous Land Vehicle in a Neural Network*, page 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016.

[17] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel : Model-based offline reinforcement learning. *CoRR*, abs/2005.05951, 2020.

[18] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: model-based offline policy optimization. *CoRR*, abs/2005.13239, 2020.

[19] Manfred Morari, Carlos E. Garcia, and David M. Prett. Model predictive control: Theory and practice. *IFAC Proceedings Volumes*, 21(4):1–12, 1988. IFAC Workshop on Model Based Process Control, Atlanta, GA, USA, 13-14 June.

[20] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *CoRR*, abs/2008.05556, 2020.

[21] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, jul 1985.

[22] Anton Francois. *Voronoi diagrams of semi-algebraic sets*. PhD thesis, University of British Columbia, 2004.

[23] Daniel Garcia-Castellanos and Umberto Lombardo. Poles of inaccessibility: A calculation algorithm for the remotest places on earth. *Scottish Geographical Journal*, 123(3):227–233, 2007.

[24] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

6.1 Referencing to literature

APPENDIX A: DATA

If appendices are necessary, they appear at the end of the document. Use ‘appsection’ instead of ‘section’.

APPENDIX B: MORE DATA

Even more data.