Deutscher Luft- und Raumfahrtkongress 2022
DocumentID: 570497

Emergency Pilot: Automated Flight Guidance After a Loss of Thrust Based on Deep Reinforcement Learning

Walter Laurito [1,*], **Renè Titze** [2] **and Wolfram Schiffmann** [2]

[1] *FZI Research Center of Information Technology, Karlsruhe, Germany*
[2] *Faculty of Mathematics and Computer Science, FernUniversität in Hagen, Germany*

Correspondence*:
Walter Laurito
Laurito@fzi.de

ABSTRACT

The fatality rate for flights involving light, fixed-wing aircraft is relatively high, and most accidents occur after a loss of thrust in the lower airspace. In these emergency situations, with a lack of potential energy, an assistance or automated system could lead the aircraft safely to an appropriate landing field. Some solutions for path planning and guidance exist, however most of them rely on a simplified model of the environment and the aircraft's dynamics to generate a path. Thus, those solutions tend to be rigid and less reliable during an emergency; especially in the occurrence of wind. Moreover, many solutions don't consider the expected landing direction and the heading of the aircraft has to have, when reaching the landing field. In this work, we tackled these issues by focusing on the creation of a real-time guidance system for 3D trajectory planning after a loss of thrust based on deep reinforcement learning (DRL). In DRL, an agent learns through trial and error by interacting with an environment. DRL is especially useful in uncertain environments, where many parameters can't be calculated in advance, which is the case in an emergency. Therefore, to train the agent to guide a fixed-wing, engines-off aircraft to an arbitrary target position, we developed and implemented multiple simulation environments. Furthermore, we incorporated wind into one of these environments. The created software package of the environments can be found online [1]. Usually, complex calculations are needed to model the engines-off flight dynamics and to generate a 3D path (guidance) under wind. With DRL these calculations can be avoided. By using shaped reward functions, we trained a neural network to successfully select directions and glide angles to lead the aircraft to an arbitrary, chosen landing field in real-time while avoiding accidents. The success rate during our experiments was high: In most cases, the aircraft reaches the target position from the correct direction and with the expected heading.

Keywords: Deep Reinforcement Learning, Aviation, Flight Guidance, Emergency Landing, Trajectory Planning

1 INTRODUCTION

Compared to commercial aviation, the fatality rate for flights involving light, fixed-wing aircraft is still relatively high (*Air safety statistics in the EU - Statistics Explained* 2020), where many accidents occur after a loss of thrust (Dorr 2018). After a loss of thrust, the pilot has only a short amount of time to find a safe path to a nearby landing field. The pilot has to consider multiple things in parallel: The remaining

[1] Flight guidance env `https://github.com/lauritowal/guidance_flight_env`

doi: 10.25967/570497

altitude to the ground, other passengers, the aircraft's velocity and probably even wind. Particularly inexperienced pilots might be overwhelmed by the fast-paced emergency situation during the flight. Instead of the inexperienced pilot, an automated system could lead the aircraft to a close landing field and ideally prevent fatal accidents.

Such an automated system could be structured as a Guidance, Navigation and Control (GNC) system. The guidance system, part of the GNC system, is responsible for generating a path to a specific target, a near landing field. The guidance system obtains the aircraft's location, velocity and attitude from a navigation system. From the obtained information, the guidance system generates the path and provides desired directions (headings) of the aircraft to a control system. Then, the control system leads the aircraft to follow these directions and eventually the aircraft moves along the path. This work focuses mostly on the creation of a real-time guidance system. The guidance system generates a 3D path that leads a fixed-wing aircraft after a loss of thrust to a near landing field.

Reinforcement Learning (RL) is a subfield of Machine *Learning* (ML), which is well suited for uncertain, changing environments. In RL, an agent learns directly by interacting with an environment. After learning, a successful agent is able to generalize (up to a certain point) to a newly presented situation by the environment. This means, the agent is capable of selecting the optimal action for a given observation at each time step. The combination of deep neural networks (Goodfellow, Bengio, and Courville 2016) with reinforcement learning led to the term *Deep Reinforcement Learning (DRL)*. In the past few years, DRL was applied to a variety of complex problems and demonstrated remarkable achievements, for example, in the field of robotics (Akkaya et al. 2019; Gu et al. 2016), digital games (Mnih et al. 2013; Berner et al. 2019 and board games (Schrittwieser et al. 2020; Vinyals et al. 2019). Using DRL for flight guidance has the following advantages:

- Although the training phase of an agent can be relatively long, after the training phase is completed, the agent can be used in real-time. The trained agent does not need to perform computationally expensive operations to generate a path. During the flight, decisions for the aircraft's next direction can be made nearly instantly.
- A trained DRL-agent could probably perform better than humans and even reach superhuman performance, as shown for example in (Mnih et al. 2013) and (Silver et al. 2017). Therefore, when combined with a conventional control system, DRL could be ideal for guidance in an emergency, since a trained agent could reach a landing field where even a human expert pilot would fail.
- There is no need to have a model of the aircraft, neither of the surroundings. Therefore, the same approach must be used to train an agent on different aircraft and environments.

There are works applying DRL to path planning in aviation. However, to our knowledge, no work exists that uses DRL to specifically address the presented problem above: Generating a 3D path to a landing field, for a fixed-wing aircraft in case of total loss of thrust. The exceptional achievements in the field of DRL and the described advantages to flight guidance, led to the decision of the authors of this work, to study the potential of DRL applied to the described problem above.

In this work, a trained DRL agent acts as the guidance system. To train the agent, three custom OpenAI Gym (Brockman et al. 2016) environments were developed. Those environments were created with a typical emergency situation in mind: Low remaining altitude, loss of thrust, and additional wind. After the training, the agent was evaluated in different experiments. Overall, in many cases, the agent guided the aircraft successfully to the target. In one of the experiments, it achieved a high mean success rate of 73 %.

For the training, the light Cessna172P aircraft was selected inside the simulation. The reason for this selection is stated at the beginning of this section: The fatality rate for flights involving light, fixed-wing aircraft is still relatively high (*Air safety statistics in the EU - Statistics Explained* 2020). Yet, since model-free DRL is used, it is not necessary for the DRL-agent to receive information about the specific aircraft in advance. The agent learns about the aircraft dynamics from experience during the training phase. This has the advantage that the methods developed in this work can easily be extended and applied to other types of aircraft, by simply using a different aircraft model during training.

The main contribution of this work is a first insight into the applicability of DRL to create a flight emergency guidance system for fixed-wing aircraft. This guidance system is responsible for generating a 3D path to a landing field after a loss of thrust. Furthermore, the developed and implemented environments used to train and evaluate the DRL agent can be used for future research by others.

The remainder of this paper is structured as follows: Section 2 states the problem in more detail. Section 3 discusses related works. Section 4 introduces fundamentals and background knowledge. Section 5 presents the methods. Section 6 contains the experiments with the setup. Section 7 presents the results, and Section 8 the discussion. Finally, the last Section 9 provides the conclusion and an outlook for further work.

2 PROBLEM STATEMENT

In this work, the goal is to study the potential of deep reinforcement learning applied to the creation of a flight emergency guidance system. A DRL agent acts as the guidance system and shall have the following capabilities:

- Generating a 3D path from the aircraft position to an arbitrary target position. This is done by the guidance system communicating the desired direction (heading) to the control system in each time step.
- Considering that the initial configuration of the aircraft is arbitrary. This includes the heading, altitude, position, etc.
- Guiding the engines-off aircraft (loss of thrust) to reach the target with the expected heading. During the last part of the landing procedure, the final approach, the aircraft needs to turn into line with the runway to be able to proceed to the round-out stage smoothly (Crocker 2007). The round-out stage, at around 15 ft above the runway, is the last phase just before touchdown (United States Department of Transportation 2016; Allerton 2009, p. 8–6.). In a normal situation, during the round-out stage, the nose of the aircraft is raised for some time to reduce the rate of descent before touching the ground (Allerton 2009, p. 194). Therefore, to be able to proceed to the round-out stage smoothly, a major weight shall be given to the following: The aircraft must end the final approach with the expected heading, which includes the fact that the aircraft needs to reach the landing field from the correct side, as can be seen in Figure 1. Reaching the landing field from the correct side is especially important for emergency situations, since obstacles could be found on the other side of the landing field. Moreover, the influence of existing wind on the aircraft affect the choice of the particular direction of the landing field ((ASA) 2017, Chapter 8, p. 14).
- Leading the aircraft also under the influence of constant wind, since wind can affect the flight path significantly. Wind speed in the range of 0 to 3000 ft/min (Beaufort scale 1-7) shall be considered.

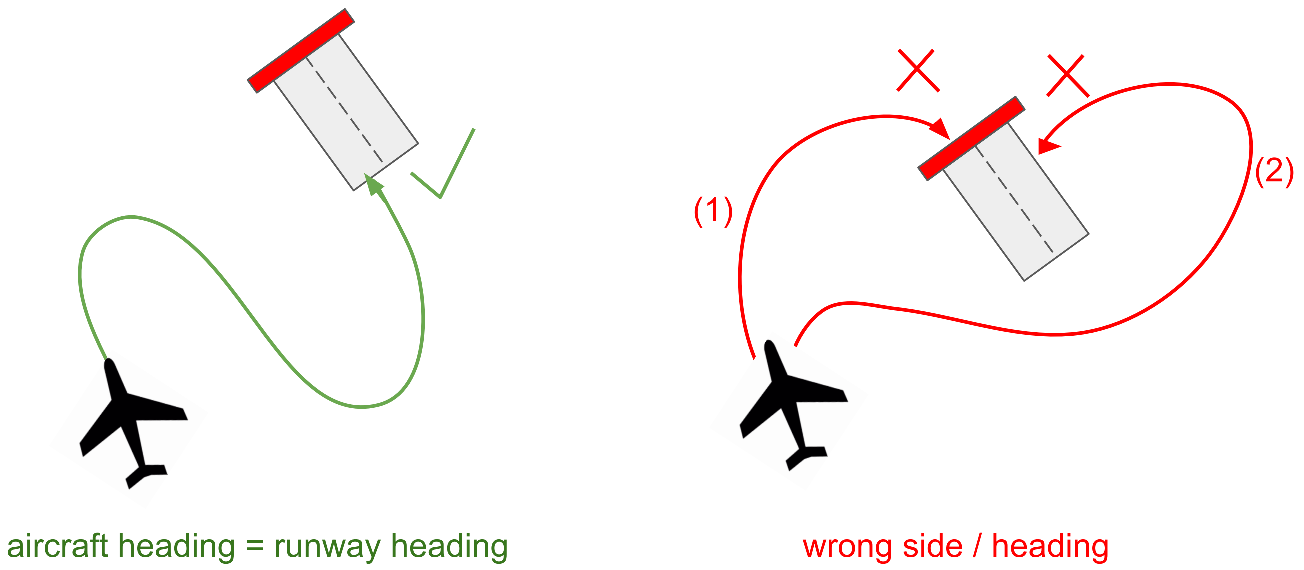


Figure 1. The image on the left shows a correct path to reach the runway with the expected heading. The image on the right shows two incorrect paths, where the aircraft would reach the runway with a high heading error. Here, the red rectangle at the top of the runway indicates the wrong side for landing.

For the agent to obtain the above capabilities, it needs to be trained. Therefore, multiple environments need to be created to train the agent in simulation.

In addition to the guidance system, a control system is essential, which receives the guidance system's instructions (desired heading and pitch angle). Based on those instructions, the control system is responsible to output necessary commands to correct the aircraft's orientation. Furthermore, the control system is used for maintaining the *best glide speed*. The best glide speed, is the speed that allows the aircraft to travel the greatest forward distance for each increment of altitude lost (Administration 2004). For the Cessna172P, the best glide speed is at around 65 Knots-Indicated Air Speed (KIAS) (*Best Glide Speed and Distance* 2018). Only a minimal control system shall be created, since the control system is not the focus of this work.

A navigation system is assumed to be available, which is part of many aircraft, and also available in the simulation in this work. Usually, the navigation system obtains information about the aircraft's location, velocity and attitude from the aircraft's sensors and provides this information to the guidance system Allerton 2009, p. 247.

Success is defined by the aircraft's distance to the target and the heading error at the end of the episode. Furthermore, the difference in altitude of the aircraft to the target position needs to be just above ground level.

The landing field can be a conventional runway or any other quite flat area which can be used for an emergency landing (for example, a grass field).

Guiding the aircraft through the round-out stage and touchdown phase is not part of this work. Moreover, considering obstacles on the flight path is left to future research.

3 RELATED WORKS

Klein et al. have a proposed the Emergency Landing Assistant (ELA) for an aircraft after a loss of thrust (Klein, Klos, Lenhardt, et al. 2018; Klein, Klos, and Schiffmann 2020). There, Dubins paths were generated to obtain a flight path from an arbitrary aircraft's configuration (position, heading, etc.) to an emergency landing field, while considering constant wind. The main idea of their approach was the following: Moving the landing field's threshold opposite to the wind direction by a length of the wind speed times the glide time. This created a new virtual target position. Next, by following the path to this virtual threshold instead, the aircraft would be compensating for the displacement caused by the constant wind. Eventually, the aircraft would then reach the threshold of the real landing field on the ground. Klein et al. have inspired the author of this thesis to create an alternative solution to ELA, based on DRL. Stephan et al. also used Dubins curves successfully to generate 3D paths to an emergency landing field, however they did not consider constant wind (Stephan and Fichter 2016).

The presented works demonstrated solutions for generating paths based on simple Dubins curves, however some shortcomings still exist. First, the radius r of the turns (L, R) needs to be provided in advance. The turn radius depends on the aircraft's properties and current flight dynamics (forward speed, turn speed, roll angle, etc.). For this reason, a model of the aircraft is needed to select the correct radius. Hence, the aircraft's model has to be quite realistic to ensure that the aircraft in emergency can actually carry out the required maneuvers needed to turn and follow the path. This also implies that, inconveniently, for each aircraft a different model is required. Furthermore, the necessity to provide the radius in advance for the path generation also reduces available maneuvers, because the radius can't be changed during an aircraft's ongoing turn operation. Moreover, having a fixed radius restricts the aircraft to turn with the same roll angle (bank angle) and velocity. This means, that unexpected perturbations (e.g., obstacles, changes in wind speed or direction, pilot errors, etc.) could render a generated path useless. Klein et al. tried to solve this by recalculating new paths during the flight (Klein, Klos, and Schiffmann 2020), however it could still be impossible following these generated paths. Furthermore, the bank angle and speed affects the aircraft's available remaining altitude and time before reaching the ground. Therefore, during an emergency, it is of major importance to select an adequate radius. Using a fixed radius for all turns however could mostly result in a compromise and not in the optimal solution for a specific situation. Another shortcoming in (Klein, Klos, Lenhardt, et al. 2018) is that in some cases the position of the virtual landing field can't be calculated directly and needs to be approximated. This could lead to the aircraft not terminating exactly at the correct position of the actual landing field.

As described above, by creating a solution based on deep reinforcement learning no model in advance is needed, which eliminates the problem of creating a model and specifying its parameters. Therefore, the same approach based on DRL could easily be applied for training models using different aircraft. Furthermore, setting a turn radius in advance is not needed, since a trained DRL agent can decide to adjust the turn radius as needed during the flight. The path is generated in real-time and not beforehand, which makes the DRL solution more flexible. The DRL agent would be trained on different scenarios to be able to generate an optimal path for a specific situation.

Apart from approaches based on Dubins paths, there are other solutions. One of them is (Váňa et al. 2018). Váňa et al. proposed a solution based on Rapidly-exploring Random Trees (RRT*). There, trajectories to potential emergency landing fields to the aircraft's location were generated and collected at any time during the flight. In case of a loss of thrust, a path to a landing site was then already available and thus needed not to be generated via the computational expensive RRT* algorithm. However, there were cases where not enough trajectories could be generated in time, for example in the case of a loss of thrust directly after take off. Furthermore, small changes occurring during the flight were not taken into account. The authors did not consider wind, for example, which can have a substantial influence on the quality of a selected path.

Most literature on using deep reinforcement learning for flight guidance focused on path planning in situations where the aircraft's engines were fully functional. As described before, no previous work was found that used DRL to specifically address the presented problem above: Generating a 3D path to a landing field for a fixed-wing aircraft after a loss of thrust. Nonetheless, there are of course works which addressing the problem of path generation for aircraft in general. Those used a solution based on DRL. Some are shortly introduced.

In (Z Wang et al. 2018), a simple Deep Q-Network (DQN) was used to train an agent model for guidance, leading the fixed-wing aircraft to a landing field. During the training, the agent was rewarded for landing successfully and penalized for leaving a specific sector or if running out of fuel. However, the approach was simplified to the 2D space. Moreover, the occurrence of wind was not investigated.

In (Yan, Xiang, and C Wang 2019), the authors used a Dueling Deep Q Network (D3QN) algorithm in a dynamic environment to train an Unmanned Aerial Vehicle (UAV) for path planning while avoiding static and moving obstacles. The paper did not provide a solution in a situation with wind. Furthermore, the simplification of the environment to a 2D image map makes it difficult to apply the solution during a real-world emergency landing.

In (Xi and Liu 2020), the authors addressed the problem of path planning to a specific target for an UAV in dynamically changing environments with obstacles. They proposed a training scheme for reinforcement learning based on the Deep Deterministic Policy Gradient (DDPG) algorithm. The purpose of the scheme was to train the agent such that the UAV behaved in accordance with the real human intent. However, the presented solution was only based on a 2D environment and did not consider wind.

Bouhamed et al. (Bouhamed et al. 2020) used the technique of transfer learning to train an agent model with DDPG to lead an UAV to a static or moving target. As in the case of many other methods mentioned before, no wind was considered in the proposed solution.

Apart from works directed to aviation, two other works are shortly presented, which inspired the use of the cross track error in this work. Both following works applied DRL to the *path-following problem*, where a vehicle needs to reach and follow a *predefined* path. To measure if a vehicle is following the path, the cross track error is calculated. The cross track error is the normal from a vehicle to a specific path. The objective is to reduce the cross track error to zero. By doing this, the vehicle approaches the path and stays on it:

In (Martinsen and Lekkas 2018b) DRL was applied to the path-following problem for a mariner-class vessel. They showed that the trained agent was able to minimize the cross track error to a straight-line path also under the presence of ocean currents.

In (Havenstrøm, Rasheed, and San 2021), DRL was applied to the path-following problem for an underwater vehicle. Apart from the horizontal cross track error, they calculated the along-track error and an additional error to the path, which they called vertical track error. The additional vertical track error allowed the agent to follow the path also in the three-dimensional space.

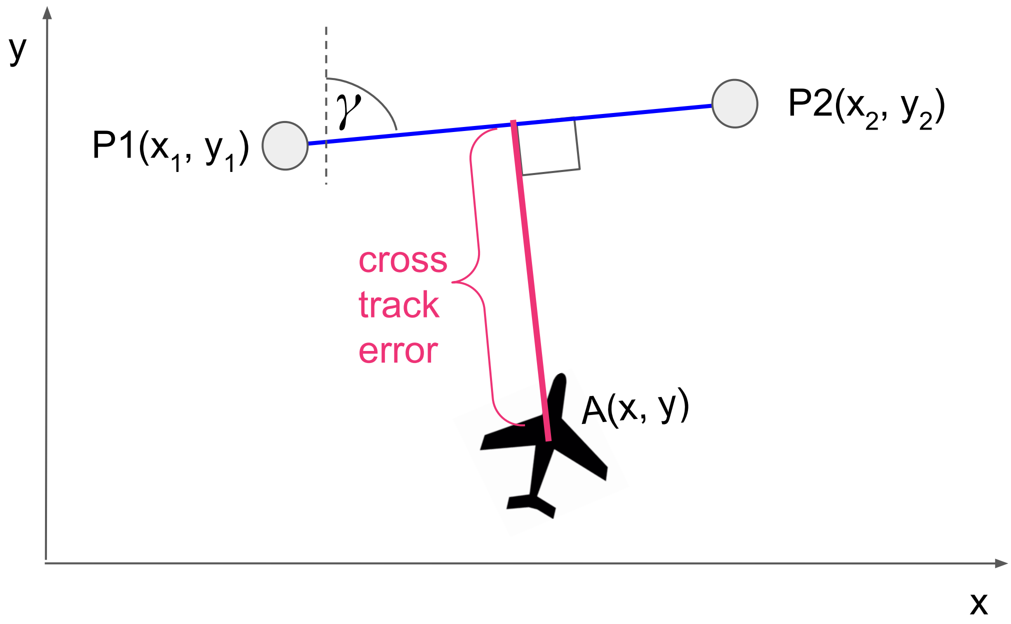


Figure 2. The cross track error

4 FUNDAMENTALS

4.1 Flight Dynamics Model Aircraft

Flight Dynamics Model (FDM) is used for representing an aircraft in the simulated environments. The (FDM) is the mathematical model that determines the physics of a flying aircraft in simulation. It is therefore responsible for providing the aircraft's equations of motion and calculating the forces and moments acting upon the aircraft. In this work, the widely used JSBSim library was applied as FDM with the included Cesna127P aircraft (J Berndt 2004; Perry 2004; J Berndt and De Marco 2009; J S Berndt et al. 2011). The method should work also when training the agent with other types of aircraft, since it is based on model-free DRL.

4.2 Track Errors & Path Following

In this section, the principles of the *cross track error* and *vertical track error* are described, since they are used to follow a path in the created environments. In a 3D Cartesian coordinate system, consider a straight line path in the XY-plane between two points $P1(x_1, y_1)$ and $P2(x_2, y_2)$, as can be seen in Figure 2. Furthermore, the angle γ of the path and the position of the aircraft at the point $A(x, y)$ are given. The cross track error e_{cross} is the normal from the aircraft to the straight line path and is calculated as shown in Equation 1, as described in (Martinsen and Lekkas 2018a):

$$e_{\text{cross}} = -(x - x_1)\sin(\gamma) + (y - y_1)\cos(\gamma) \tag{1}$$

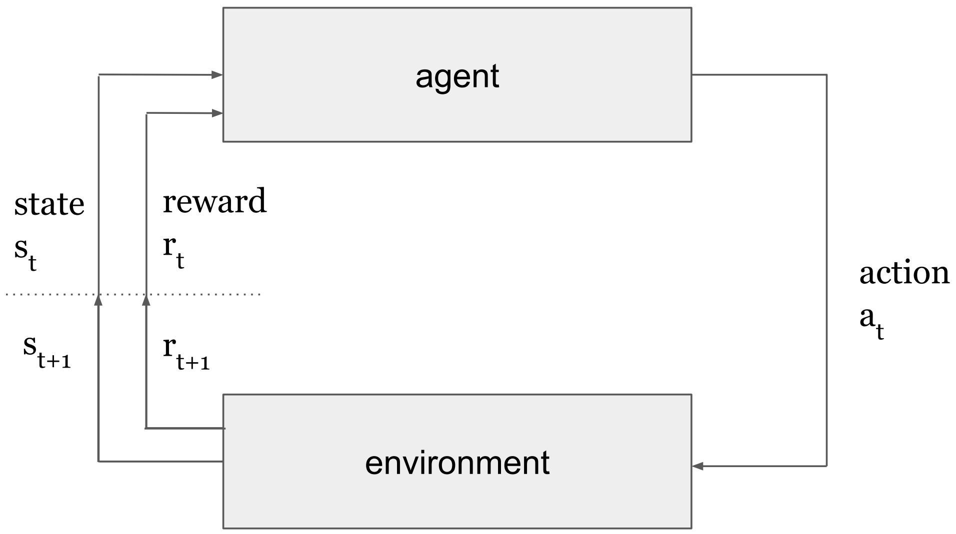


Figure 3. The interaction of the agent with the environment

In this work, the vertical track error is obtained by using a similar calculation as for the cross track error. The only real difference is that the path lies in the YZ-plane instead:

$$e_{\text{vertical}} = -(y - y_1)\sin(\gamma) + (z - z_1)\cos(\gamma) \tag{2}$$

The angle γ is always known in this work (the runway heading or approach slope), therefore there is no need to calculate it from the two points P1 and P2 first.

4.3 Reinforcement Learning

Reinforcement learning allows automating decision-making by maximizing a reward signal, a simple scalar (Sutton and Barto 2018, p. 1). This is accomplished by a decision maker, called *agent*. The agent learns a specific behavior from experience, which is generated by interacting with an *environment*.

A problem in RL is often stated formally as finite horizon Markov Decision Process (MDP), which consists of the tuple (S, A, R, P). At each time step t, the agent interacts with the environment by taking an *action* $a_t \in A$, where A is a set of actions. After doing so, the environment transitions from the current state $s_t \in S$ into a next state $s_{t+1} \in S$, where S is a set of states. The transition probability function $P(s_{t+1}|s_t, a_t)$ represents the probability for that transition, given the current state s_t and the agent's chosen action a_t. Subsequently, the agent receives a reward r_{t+1} produced by the reward function $R(s_t, a_t, s_m t + 1)$. Furthermore, the agent receives the new state s_{t+1}. The state s_{t+1} becomes the current state s_t and the reward r_{t+1} the current reward r_t. Based on those received values, once more, the agent selects the next action to interact with the environment in the next time step $t + 1$. In the finite MDP, the environment terminates, when a terminal state is reached. There, the time horizon from t_0 to the last time step is called an *episode*. Tasks with episodes are called *episodic tasks*. The sequence $(s_0, a_0, r_0), (s_1, a_1, r_1), ...(s_n, a_n, r_n)$ of an episode is called *trajectory*. Figure 3 demonstrates the typical control loop of RL.

The *goal* of the agent is to maximize the received reward. However, the agent tries not to maximize its immediate reward, but the cumulative long term reward, called *expected return* G_t. The agent achieves this by taking the actions for a given state that lead to the highest cumulative reward. The function which produces the actions of the agent for a given state is called *policy* $\pi(a|s)$. If the agent follows the policy π at time step t, then $\pi(a|s)$ represents the probability for the agent to select action a given state s.

$$G_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{3}$$

where $\gamma \in [0, 1]$ is the discount rate. The discount rate influences how the agent values future rewards.

5 METHODS

5.1 Custom Environments

To apply Deep Reinforcement Learning to the problem of emergency flight guidance, three custom environments were created:

- `Default`: The `Default` environment contains no wind. Moreover, the action space has one dimension.
- `Wind`: The `Wind` environment is an extension of the `Default` environment. It is extended by adding constant wind during the training of an agent. The intensity of the wind is increased over the training time. As in the `Default` environment, the action space has one dimension.
- `TwoActions`: A third environment was created where the action space consists of two dimensions. There is no wind during the training in the TwoActions environment.

Our software `guidance_flight_env`, which provides the custom environments, started as a git fork of Gordon Rennie's open-source software *Gym-JSBSim* (Rennie 2018), but was adapted for flight guidance.

5.2 Main Components of the Custom Environments

In this section, the most important components of the custom environments and the interaction with the agent are described. The interaction is illustrated in Figure 4.

As reported above in Section 4, the agent interacts with the *environment* by selecting an action a_t from the environment's action space A at each time step t. For the `Default` environment and the `Wind` environment the action a_t is a simple angle value, the desired heading, between 0° and 359°. After the agent has selected the heading, it is passed to the environment, where it is then internally processed by a heading hold PID controller. The controller, returns the aileron command necessary for pointing the aircraft to the desired angle. The aileron command is set in the Flight Dynamics Model (FDM) component, and subsequently a number of simulation steps are executed to update the FDM. Next, a new state is obtained, and the corresponding reward is calculated. Both are then passed back to the agent. In addition to the heading hold PID controller, a second controller, more specifically, a pitch angle hold controller, is used to keep also the aircraft's pitch angle in a fixed position of 0°. The reason for this was shortly described in Section 2.

5.3 Overview of Core Concept

In this section, the core concept which is common in all three environments is shortly presented. Details are then described in the proceeding sections.

A line, perpendicular to the extended center line of the runway, divides the XY-plane of the Cartesian coordinate system into two different areas: *Area 1* and *Area 2*. In the case of the aircraft's location being in

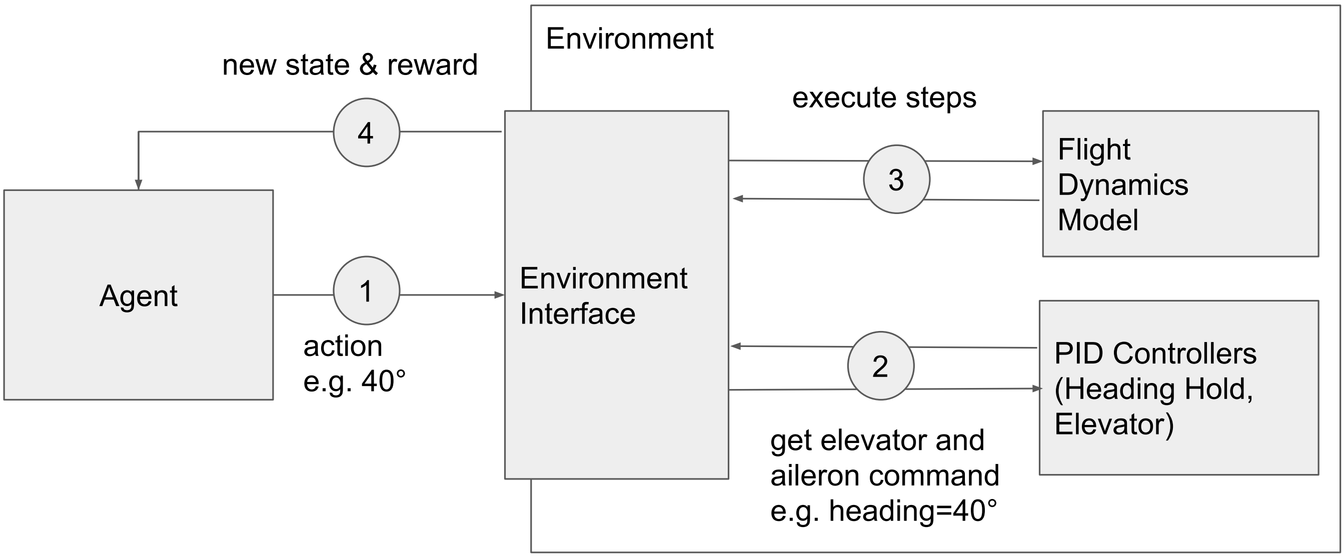


Figure 4. The interaction of the agent with the custom environments

Area 2, the cross track error e_{cross} of the aircraft to the perpendicular line is calculated and is included in the environment's state at time step t. The agent then observes this state and, while trying to maximize the reward by reducing the error, it guides the aircraft to *Area 1*. The described procedure is illustrated in Figure 5.

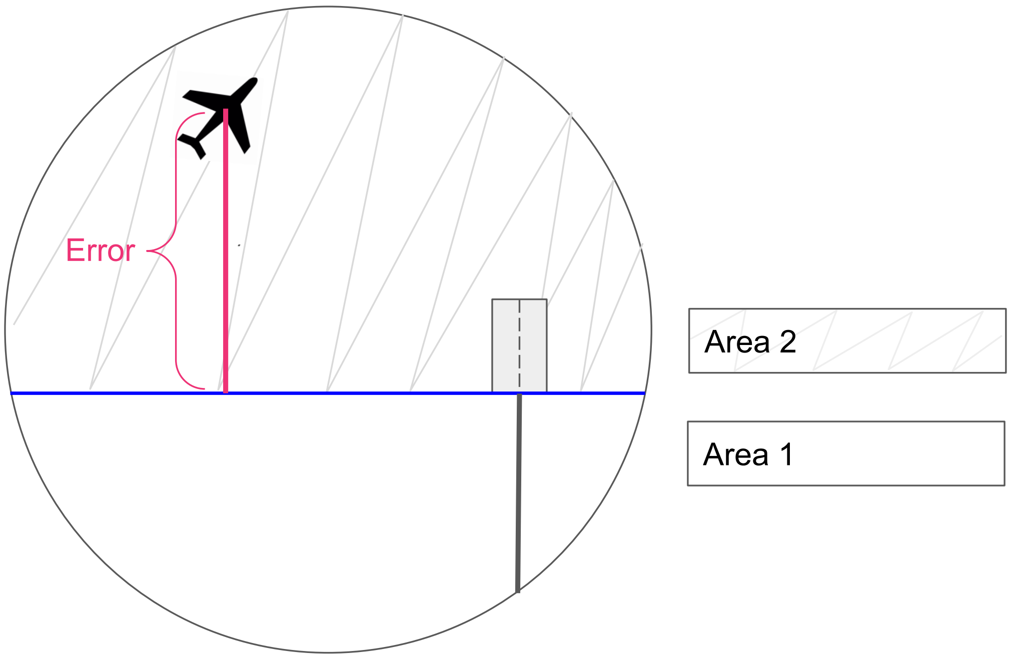


Figure 5. The aircraft is in *Area 2*. The cross track error to the perpendicular line (blue) to the extended center line of the runway (black) is calculated.

Once the aircraft is in *Area 1*, the cross track error to the extended center line of the runway is added to the state instead. Now, the agent is incentivized to guide the aircraft to the target position. See Figure 6

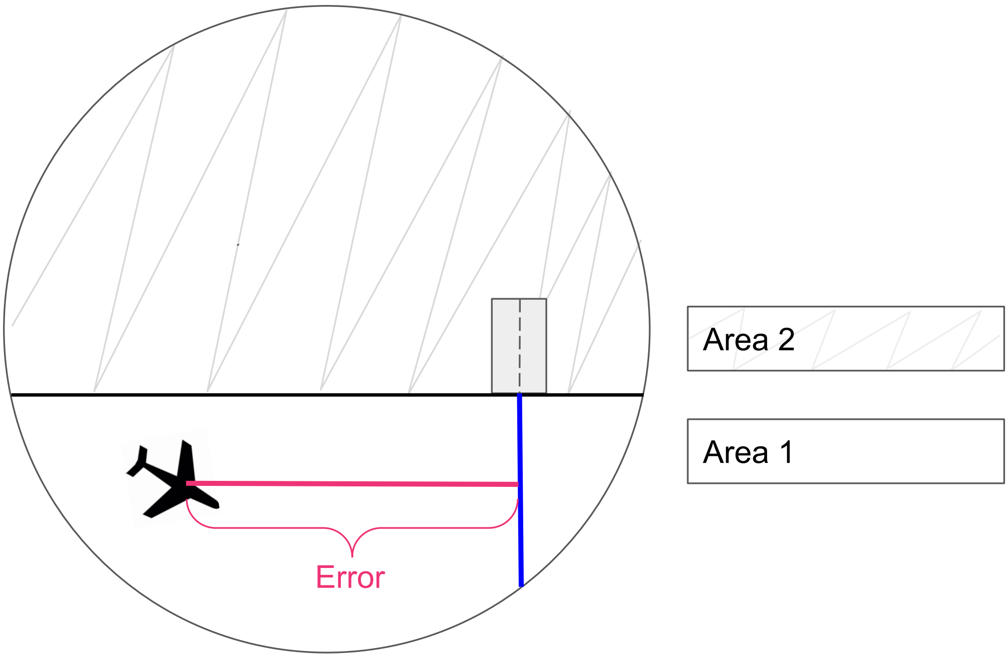


Figure 6. The aircraft is in *Area 1*. The cross track error to the extended center line of the runway (blue) is calculated and added to the state.

Similar to the cross track error in the XY-plane, the error e_{vertical} in the YZ-plane is calculated and added to the state. The error e_{vertical} is obtained by calculating the vertical distance of the aircraft to an imagined line created by a specific approach slope γ (Havenstrøm, Rasheed, and San 2021). An example is demonstrated in Figure 7

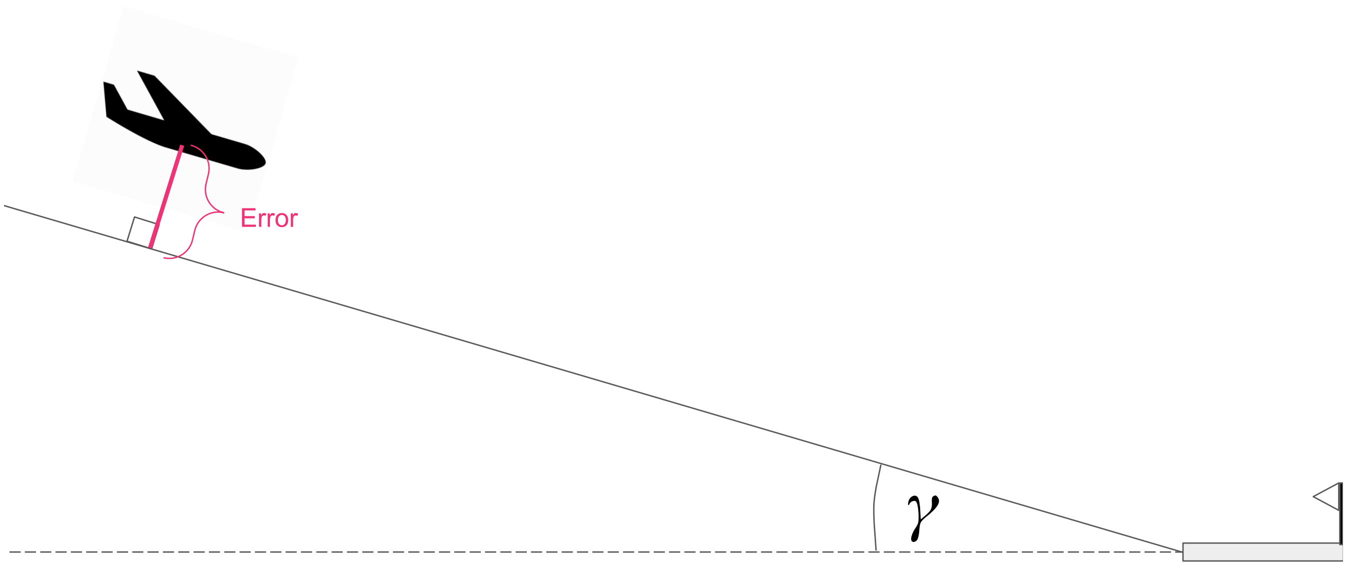


Figure 7. Here, the e_{vertical} is calculated from the aircraft's position in the YZ-plane to the line with the slope angle

The agent receives positive and negative rewards depending on the magnitude of both errors. The rewards should motivate the agent to guide the aircraft closer to the target, while maintaining the correct heading and approach slope angle. When the aircraft reaches the target, the episode terminates. It does also terminate when the agent has no remaining altitude left before achieving the target.

5.4 Angles and Discontinuity: TwoActions-Environment

For humans, it is mostly clear that an angle of 359° is relatively close to an angle of $1°$. However, for a neural network this relation seem to be hard to learn, since discontinuity occurs between 2π (360°) and 0 (Zhou et al. 2019); numerically the values 360 and 0 are very far apart. Therefore, the following assumption is made: Using angle values in the action and observation space could introduce noise during the training process, which would lead to worse performance of the RL agent.

To avoid this noise, some researchers, (Berner et al. 2019, Xi and Liu 2020), encode an orientation angle value α as shown in Equation 4.

$$\alpha \rightarrow (\sin(\alpha), \cos(\alpha)), \text{where } \alpha \in [0, 2\pi]) \tag{4}$$

They then used the tuple of $(\sin(\alpha), \cos(\alpha))$ in the observation- or action space to represent one angle. See Figure 8. This implies that the neural network only received the tuple, instead of simply a single angle value.

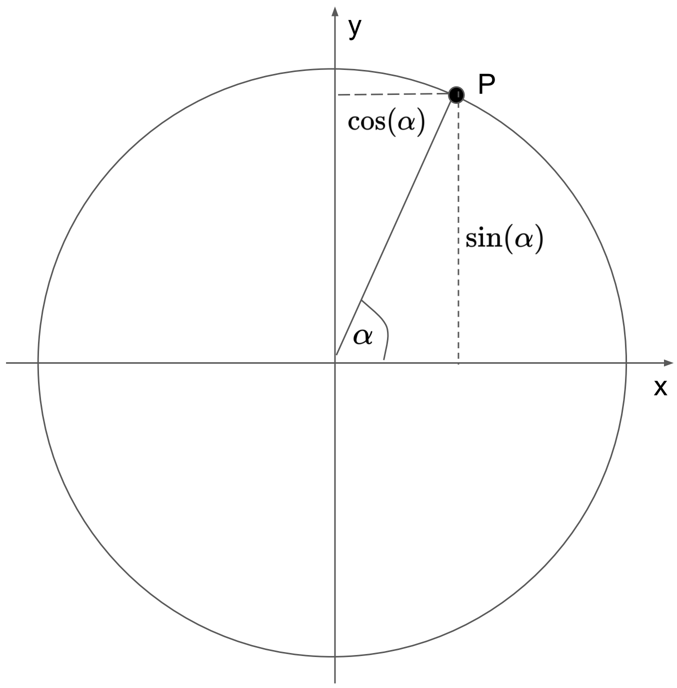


Figure 8. An angle α can be represented by the two values: $\sin(\alpha)$ and $\cos(\alpha)$

To test the above assumptions, the `TwoActions` environment was created and there all angles in the state and action space were encoded as in Equation 5. To obtain the actual angle again, both values were decoded in the environment using the $arctan2$ function as shown in Equation 5

$$(\sin(\alpha), \cos(\alpha)) \rightarrow arctan2(\sin(\alpha), \cos(\alpha))), \text{where } \alpha \in [0, 2\pi] \tag{5}$$

The `Default` environment, on the other hand, uses the single angle values directly. Later, in Section 6 the two environments are compared by demonstrating the performance of the trained agent in both.

5.5 Initial Setup

At the beginning of each episode, the following steps are executed:

1. **Cartesian Coordinates & Aircraft's Position:** The altitude and geographical coordinates (latitude, longitude) of the aircraft are converted into coordinates of the 3D Cartesian coordinate system (x, y, z). This conversation simplifies calculations, since now simple vector algebra can be used. The conversation is justified, since the resulting conversation errors are kept to a minimum when distances between objects are relatively low, which is mostly true in the case of an engine's failure. After the conversation, the origin of the XY-plane (x=0, y=0) represents the aircraft's initial x, y coordinates. Next, the aircraft's altitude (z coordinate) is chosen arbitrary between a minimum and maximum value. Now, the target position and all future positions of the aircraft are calculated relative to the origin. Using relative positions, instead of absolute positions, improves the generalization capacities of the agent's neural network (ibid.).

2. **Aircraft Heading:** The heading of the aircraft is chosen arbitrary in the range of $[0, 2\pi]$.

3. **target position:** After setting the origin to the aircraft's initial position, the position of target position is chosen arbitrary in a specified radius from the origin.

4. **Landing Field's Heading:** The heading of the landing field is kept at 0° for all episodes during the training. This simplification reduces training time, since the agent does not need to learn to adapt to different targets. However, this simplification does not restrict the capabilities of the agent. A simple offset to all angles can be added when using the agent after the training, as can be seen in the example in Figure 9. The value of the offset is the real heading of the landing field. Since all distances are not absolute but relative, after adding the offset everything works as it would for the case of the 0° headings during the training.

5. **Approach Slope:** The approach slope of the target landing field is set to a fixed value of 4° for all episodes, since approach slope angles of most runways are around 3°-5.5° Allerton 2009, p.193.

6. **Engines Off:** The simulated aircraft's engines are turned off to simulate an emergency setting.

7. **Curriculum Learning:** The technique of curriculum learning (Florensa et al. 2017) is used to train the agent. Curriculum learning was incorporated into the environment by dividing the training into five phases. When the agent's mean reward surpasses a certain threshold, then the phase number is incremented, which in turn increases the environment's difficulty. The difficulty is specifically increased by extending the radius of the circle, where the target positions are arbitrarily generated. In addition, in the `Wind` environment, the wind speed is changed in each phase.

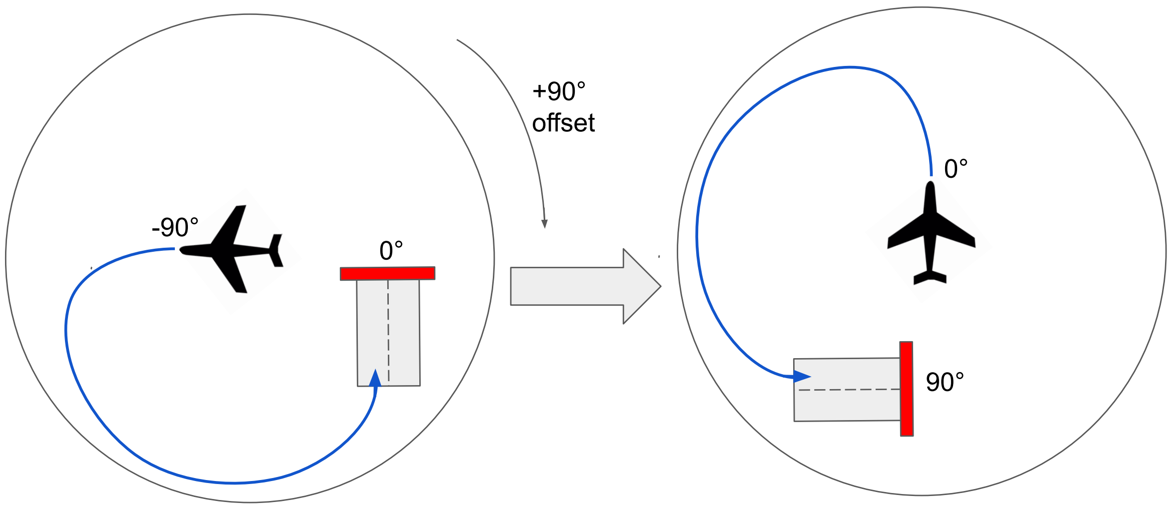


Figure 9. An example of a landing field with a real heading of 90°: After the training, adding the offset (real heading of the landing field) of 90° to all angles in the environment, allows the agent to guide the aircraft to the landing field.

5.6 Action Space

To control the flight path of the aircraft, the agent needs to be able to select the desired heading ψ of the aircraft. The aircraft's pitch angle is fixed at 0° and is therefore not part of an action.

In the case of the default and `Wind` environment, the action space has one dimension. In every time step t, the agent selects a single action a, which represents the angle value for the heading. The action could be a value ranging from $0 - 2\pi$. However, it is good practice to rescale actions to be in the range of $[-1, 1]$ (Raffin et al. 2019). The agent's selected value is then rescaled again back into the range of $[0, 2\pi]$ inside the environment.

In Section 5.4, we've made the assumption that to avoid problems with discontinuity, the desired angle ψ should be encoded using the function in Equation 4. Therefore, to the test the action a in the `TwoActions` environment, is formulated as a tuple of two values $a = [x, y]$, where x, y $\in [-1, 1]$. Inside the environment, the actual angle ψ is then obtained by $\psi = arctan2(y, x)$

5.7 State Space

5.8 State Space of the Default environment

In the case of the `Default` environment, the state consists of the following 13 elements, shown in Table 1.

The state elements are described in detail in the following subsections.

State Variable	Symbol	Unit
cross track error	e_{cross}	km
vertical track error	e_{vertical}	km
in area	a	-
difference x	x_{diff}	km
difference y	y_{diff}	km
difference z	z_{diff}	km
distance to target	d_{3D}	km
remaining altitude	h	ft
descent rate	u	ft / s
turn rate	tr	rad / s
true airspeed	v_{tas}	ft / s
true heading	ψ	rad
heading error	e_{ψ}	rad

Table 1. The state space of the `Default` environment

5.9 State Space of the Wind Environment

The state space of the `Wind` environment includes all elements from Table 1. Furthermore, the following elements from Table 2 are added.

State Variable	Symbol	Unit
total wind north	w_n	ft / s
total wind east	w_e	ft / s
drift	λ	rad

Table 2. The additional state elements of the `Wind` environment

A total constant wind originating in the East or West is added to the state of the `Wind` environment. Furthermore, the drift angle λ is added. The drift angle is the difference between the aircraft's true heading and the track angle. The track angle is the angle of the lateral track the aircraft actually flies over the ground. Under the occurrence of wind, the track angle and aircraft's true heading differ by a relatively high drift value λ, whereas in the case of no wind they should be equal. Figure 10 demonstrates this relationship.

The total wind values and drift, allow the agent to adjust the heading to react to the wind accordingly.

5.10 State Space of the TwoActions environment

The only difference in the `TwoActions` environment compared to the `Default` environment is that all the angles in the state are represented with two values $(sin(\alpha), cos(\alpha))$ as described in Section 5.4. For this reason, the true heading and the heading error are replaced by two values each. This leads to four additional elements in the state space of the `TwoActions` environment, raising the number of state elements to 17.

5.11 Terminal States

A terminal state is reached when one of the following is true:

- The aircraft's remaining altitude is lower than zero. This happens when the aircraft's z-coordinate value is lower or equal to the target's z-coordinate value.

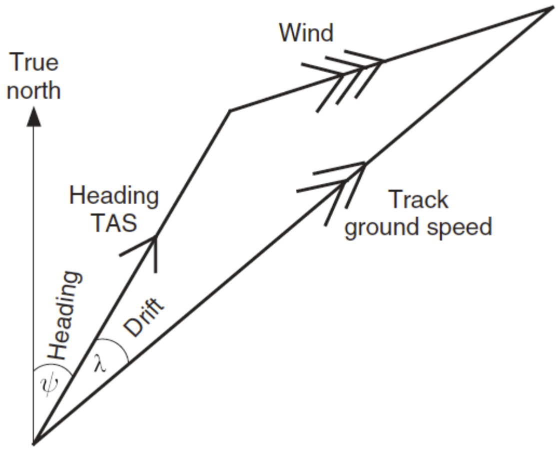


Figure 10. The aircraft's real track angle over the ground differs to its heading by the drift angle λ. (Figure adapted from Allerton 2009, p.250.)

- The aircraft reaches the target position. This is true, when the Euclidean distance from the aircraft's position to the target position is lower than the threshold $\theta_{\text{target}} = 0.1$ km. The value 0.1 was selected during preliminary experiments. A lower value for θ_{target} could have been selected, but it would have increased training times. However, except for longer training times, it is expected that the performance of the trained agent would not decline. Nonetheless, using lower values for θ_{target} should be studied in future research.
- The remaining time t is equal to zero. It should be noted that, in all experiments of this work, the altitude is chosen such that the engines-off aircraft reaches the ground before t reaches the value of zero. Therefore, in this work, the remaining time t was mostly ignored. However, it could be considered in future research, if an aircraft with temporary working engines is investigated or if a higher remaining altitude is chosen.

When one of the terminal states is reached, the episode ends and the agent is rewarded with sparse rewards as described in Section 5.12

5.12 Reward

A reward function was developed to motivate the agent to behave in the following way: Leading the aircraft to the target position in the 3D space. The target position should be reached with the expected heading. The reward function was constructed by combining sparse and dense rewards: $r = r_{sparse} + r_{dense}$.

In (Henderson et al. 2018), Henderson et al. showed that reward clipping or reward scaling effected learning of the agent significantly, when neural networks and gradient-based methods are used (which is the case in most state of the art DRL-algorithms). Therefore, all rewards in the three custom environments are kept in the range of $[-10, 10]$.

5.12.1 Sparse Rewards

In this section, the different sparse rewards, which are returned on the terminal states, are presented in detail.

Terminal State at Target: When the aircraft reaches the target from the wrong side, coming from *Area 2*, then the reward is -10. However, when the aircraft reaches the target from the correct side, coming from

Area 1, then a positive reward of 9 is given. Furthermore, the agent is rewarded for the correct heading. This is done in the following way: The heading error e_h in the range of $[x_{min}, x_{max}]$ is rescaled into the range of $[0, 1]$, where x_{max} represent the threshold (the tolerance value) for the correct heading. The value x_{min} is set to 0 and x_{max} to 10 (degrees). The scaling results in $e_{h_{\text{scaled}}}$

Then, the bonus reward is calculated in the following way:

$$r_b = 1 - e_{h_{\text{scaled}}}, \text{where } r_b \in [0, 1] \tag{6}$$

The final reward at the target terminal state is:

$$r_{target} = \begin{cases} 9 + r_b & \text{if } area = 1 \\ -10 & \text{else} \end{cases} \tag{7}$$

Remaining Altitude is Zero: When the remaining altitude is lower than zero, then the current episode is terminated. The Euclidean distance of the aircraft's position (x_1, y_1, z_1) to the target (x_2, y_2, z_2) is calculated and multiplied by a constant. The negative reward $r_{altitude}$ is calculated as in Equation 8:

$$r_{altitude} = -6 * \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{8}$$

Time is Zero: When the time step t is equal to zero, the current episode is terminated. The reward r_{time} is calculated, as in the case of remaining altitude equal to zero. (See Equation 8).

5.12.2 Dense rewards

In addition to sparse rewards, dense rewards were incorporated into all the environments to additionally encourage the agent to behave as expected. The agent receives a dense reward on each time step t. By adding dense rewards, the training time is reduced significantly.

Reach Track: As described in Section 5.3, depending on the aircraft being in *Area 1* or *Area 2*, different track errors are calculated:

- **Aircraft is in *Area 2*:** The cross track error e_{cross} is calculated as described in Equation 1 (Section 4) to the line perpendicular to the centerline of the landing field. The vertical track error e_{vertical} in *Area 2* is set to zero, which is done to motivate the agent to set focus on leading the aircraft to *Area 1* first.
- **Aircraft is in *Area 1*:** Only now, in *Area 1*, the aircraft needs to position the aircraft for the final approach, where it is important to reduce the vertical track error. Therefore, the vertical track error e_{vertical} is calculated as in Equation 2 (Section 4). Moreover, the cross track error e_{cross} is calculated relative to the extended center line of the landing field.

Next, $e_{\text{to\_track}}$ is calculated as in:

$$e_{\text{to\_track}} = \begin{cases} e_{\text{cross}} + e_{\text{vertical}} & \text{if area = 1} \\ e_{\text{cross}} & \text{else} \end{cases} \tag{9}$$

The negative reward $r_{\text{to\_track}}$ is now derived from $e_{\text{to\_track}}$ by applying the exponential function to the track as in:

$$r_{\text{to\_track}} = -\exp(e_{\text{to\_track}}) \tag{10}$$

By doing so, the negative reward increases exponentially for a larger $r_{\text{to\_track}}$. This should additionally motivate the agent to reach the track without many deviations. Finally, the resulting reward value $r_{\text{to\_track}}$ is then scaled to the range $[0, 1]$.

In addition, to $r_{\text{to\_track}}$, in each time step, the agent receives a constant negative reward $r_{\text{area2}} = -2$ when the aircraft is in *Area 2*, else $r_{\text{area2}} = 0$. Therefore, the agent receives more negative rewards when the aircraft is in *Area 2* compared to when it is in *Area 1*. Whereas, $r_{\text{to\_track}}$ acts as a helping leading signal for the agent to guide the aircraft to the track, the constant r_{area2} adds an urgency to reach the *Area 1* as soon as possible.

Keep aircraft on-track Once the agent guided the aircraft to reach the desired track, the agent then receives additional rewards to continue keeping the aircraft on that track. In this work, the aircraft is *on-track* when the following three conditions hold:

1. The medium track error e_{t_m} is lower than the threshold $\theta_{\text{on\_track}} = 0.1$. The medium track error e_{t_m} is calculated as in $e_{t\_m} = (|e_{t\_\text{last}}| + |e_t|)/2$, where e_{t_last} is the track error of the previous time step $t-1$ and e_t the track error of the current time step t.
2. The Euclidean distance in time step t is lower than the distance in the previous time step $t-1$. In both time steps, the Euclidean distance is calculated from the aircraft to the target position. This condition is important, since it ensures that the distance of the aircraft to the target is reducing in every time step.
3. The heading error e_h is lower than 90°. The reason is to only allow the agent to receive a positive reward if the aircraft reaches the target position from the correct side (from *Area 1*).

Therefore, when the aircraft is *on-track*, the agent receives a positive reward. However, to avoid positive reward cycles (or reward hacking) (Randløv and Alstrøm 1998; Ng, Harada, and Russell 1999), it is necessary to add additional negative rewards when the above *on-track* conditions do not hold anymore (i.e., when the aircraft leaves the track). For this reason, the agent receives the reward of $r_{\text{on\_track}} = 1$ if the aircraft is on-track, else the agent receives a negative reward in the form of $r_{\text{on\_track}} = -e_{\text{track\_diff}}$, where the value of $e_{\text{track\_diff}}$ is the difference of the track error e_{track} at time step $t-1$ and the current at time step t. Hence, the agent is punished for the aircraft moving away from the track and rewarded for staying on it. This is presented in the following Equation 11.

$$r_{\text{on\_track}} = \begin{cases} 1 & \text{if aircraft is on-track} \\ -e_{\text{track\_diff}} & \text{else} \end{cases} \tag{11}$$

5.12.3 Heading

When the aircraft is *on-track*, it could still occasionally oscillate horizontally in the XY-plane, for a short period of time. The oscillating is normally not a problem if the aircraft is still relatively distant. However, when close to the landing field it’s important that the aircraft does not oscillate, as is illustrated in Figure 11

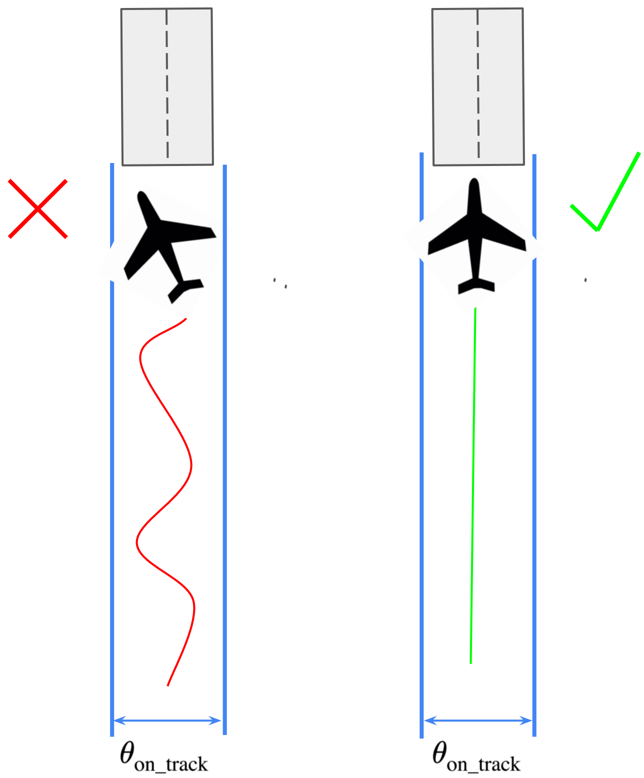


Figure 11. Given the threshold on_track, the aircraft should fly in a straight line when on-track (right side), to avoid reaching the landing field with a wrong heading (left side, intentionally shown exaggerated). To encourage the agent to achieve this, an additional reward is given.

Avoiding oscillation is necessary, to guarantee that when reaching the target position, the heading of the aircraft is as expected. Therefore, too additionally minimize the heading error, when *on-track*, the agent should be encouraged to keep the heading fixed on the desired direction.

A first idea, to avoid oscillation, could be the following: Reducing the threshold $\theta_{\text{on\_track}} = 0.1$ described in the first condition in 5.12.2 even further. However, this stricter condition would increase the difficulty for the agent to receive any positive reward ($r_{\text{on\_track}} = 1$) at all. The increased difficulty could lead to longer training times, or even to the issue that the agent does never learn to stay on-track.

For this reason, the alternative solution is to add an additional reward for the correct (or wrong) heading instead. The agent obtains the additional positive or negative reward $r_{\text{heading}} \in [-0.5, 0.5]$ only when the aircraft is on-track. Since the r_{heading} is in $[-0.5, 0.5]$ and $r_{\text{on\_track}} = 1$, the agent would, in the worst case (heading error around 90°), still receive a reward of 0.5. Therefore, the additional dense heading reward can be seen as a kind of bonus reward for the agent. Nonetheless, the agent trying to maximize the reward in general will try to maximize both $r_{\text{on\_track}}$ and r_{heading}, and therefore reaching the target with the expected heading, by avoiding oscillating.

The reward r_{heading} is obtained as follows. First, the difference $e_{\text{heading\_diff}} \in [0, \frac{\pi}{2}]$ of the heading error in time step t and the heading error in time step $t-1$ is calculated. Next, if the difference in time step t is smaller than the difference in time step $t-1$, then the reward r_{heading} with a positive value is returned, else a negative a shown in the following Equation 12:

$$r_{\text{heading}} = \begin{cases} \frac{e_{\text{heading\_diff}}}{\pi} & \text{if aircraft is on track} \\ \frac{-e_{\text{heading\_diff}}}{\pi} & \text{else} \end{cases} \tag{12}$$

Having an additional negative reward avoids positive reward cycles (Randløv and Alstrøm 1998; Ng, Harada, and Russell 1999) as described previously in Section 5.12.2.

5.13 Final Combined Reward Function

To obtain the final combined reward function, as shown in the following Equation 13:

$$r = r_{\text{sparse}} + r_{\text{dense}} \tag{13}$$

it is therefore needed to add all sparse rewards of the preceding sections, as in Equation 14:

$$r_{\text{sparse}} = r_{\text{target}} + r_{\text{altitude}} + r_{\text{time}} \tag{14}$$

Whereas the dense rewards of the preceding sections are combined, as in Equation 15:

$$r_{\text{dense}} = r_{\text{to\_track}} + r_{\text{on\_track}} + r_{\text{area2}} + r_{\text{heading}} \tag{15}$$

6 EXPERIMENTS

As described in Section 5.1 three custom environments were developed and implemented. Experiments were conducted in all three environments. In the following sections, first the setup for the experiments are described. Then the results of those experiments are presented and interpreted.

The three experiments have the following setup in common:

- **Training in the cloud:** For the training phase, two 2,25 GHz CPUs (AMD EPYC 7B12) and a GPU (NVIDIA Tesla K80 24 GB) were used.
- **Algorithm**: RLlib's Twin Delayed DDPG (TD3) was used as the agent's algorithm, since it is a current state-of-the-art algorithm and a successor of the well-studied Deep Deterministic Policy Gradient (DDPG) algorithm Lillicrap et al. 2015. Furthermore, TD3 requires less hyperparameter tuning in comparison to other algorithms like DDPG. The same hyperparameters of the TD3 were used for all experiments. The hyperparameters of the algorithm are presented in the Appendix 1.
- **Environments:** The custom environments described in Chapter 5.1 were used for training and evaluating the agent.
- **Evaluation:** After the agent was trained in one of the environments, the agent was evaluated in the same environment. For each experiment or sub-experiment, the evaluation was run thrice, each for 100 episodes. For each run, a different seed was used. The results of the three runs were then averaged.
- **Aircraft:** Since most fatal accidents in aviation occur when small aircraft are used (*Air safety statistics in the EU - Statistics Explained* 2020; Dorr 2018), the light aircraft single-engine Cessna172P was selected in the *JSBSim* FDM. Before starting the experiment, the aircraft's engine is turned off in the *JSBSim* FDM. Furthermore, throttle and mixture were set to zero. Those steps are needed to simulate broken engines and the emergency situation.

- **Initial Altitude:** The initial altitude at the beginning of each episode is selected arbitrarily in the range of 2500-3500 feet. The reasons for this altitude range are the following: Firstly, at this altitude range, only a short amount of time is left to reach the landing field. Therefore, it represents a very dangerous part of the remaining flight and simulates the real-world scenario accordingly. Secondly, a practical reason, given the relatively low initial altitude, the episodes during simulation are shorter. This led to decreased training times. Nonetheless, if needed in future research, higher numbers for the altitude range could be chosen. Except for a longer training phase, it is expected that there should be almost no difference in the agent's performance after training.

- **Curriculum Learning:** Depending on the average reward received by the agent, one of the five phases, described in Section 5.5, is selected. When entering the next phase, the target spawning range widens, as can be seen in Table 3.

Phase	Reward threshold	Target Spawn Range (km)
0	-∞	0.5
1	-900	1
2	-750	1.5
3	-600	2
4	-300	2

Table 3. The radius of the circle, where the target positions are arbitrarily generated, is adapted for each phase

6.1 Setup Experiment 1: `Default` environment

The agent was trained for 10,000 episodes (around 10 Million steps). And in addition, for 15,000 episodes (around 15 Million steps).

6.2 Setup Experiment 2: TwoActions

The agent was trained for 10,000 episodes (around 10 Million steps) in the `TwoActions` environment.

6.3 Setup Experiment 3: Wind Environment

In the `Wind` environment, during the training, the wind speed increases in each phase. See Table 4. For reducing complexity, the constant wind is set to originate in the East or West only.

Phase	Reward threshold	Speed Range [West, East] (ft/min)
0	-∞	0
1	-900	$[-600, 600]$
2	-750	$[-1200, 1200]$
3	-600	$[-2100, 2100]$
4	-300	$[-3000, 3000]$

Table 4. In the `Wind` environment, for each phase additionally the wind speed changes

The agent was trained for 10,000 episodes (around 10 Million steps).

Later, the agent trained in the `Wind` environment was evaluated under constant wind originating in the East or West. Four sub-experiments were conducted, to test the performance of the agent under the four different wind speed ranges: (1) 600 ft/min (2) 1200 ft/min (3) 2100 ft/min (4) 3000 ft/min.

7 RESULTS

7.1 Experiment 1: Default environment

Figure 12 shows the average reward for 10 Million steps. The reward curve there might indicate that the agent did not conclude its learning process, since a plateau was not yet reached. Therefore, in a second sub-experiment, the agent was trained for an additional 5 Million steps, in total for 15 Million steps. However, even at 15 Million steps, the reward curve still seemed not to have reached a plateau, which implied that further improvements might have been possible. Therefore, probably even longer training periods would have been necessary. Yet, both agents were evaluated first, and the results of the evaluations were already quite promising. Hence, the agent was not trained for longer than 15 Million steps, since the hardware resources for training were relatively limited. For future research, training periods should be prolonged.

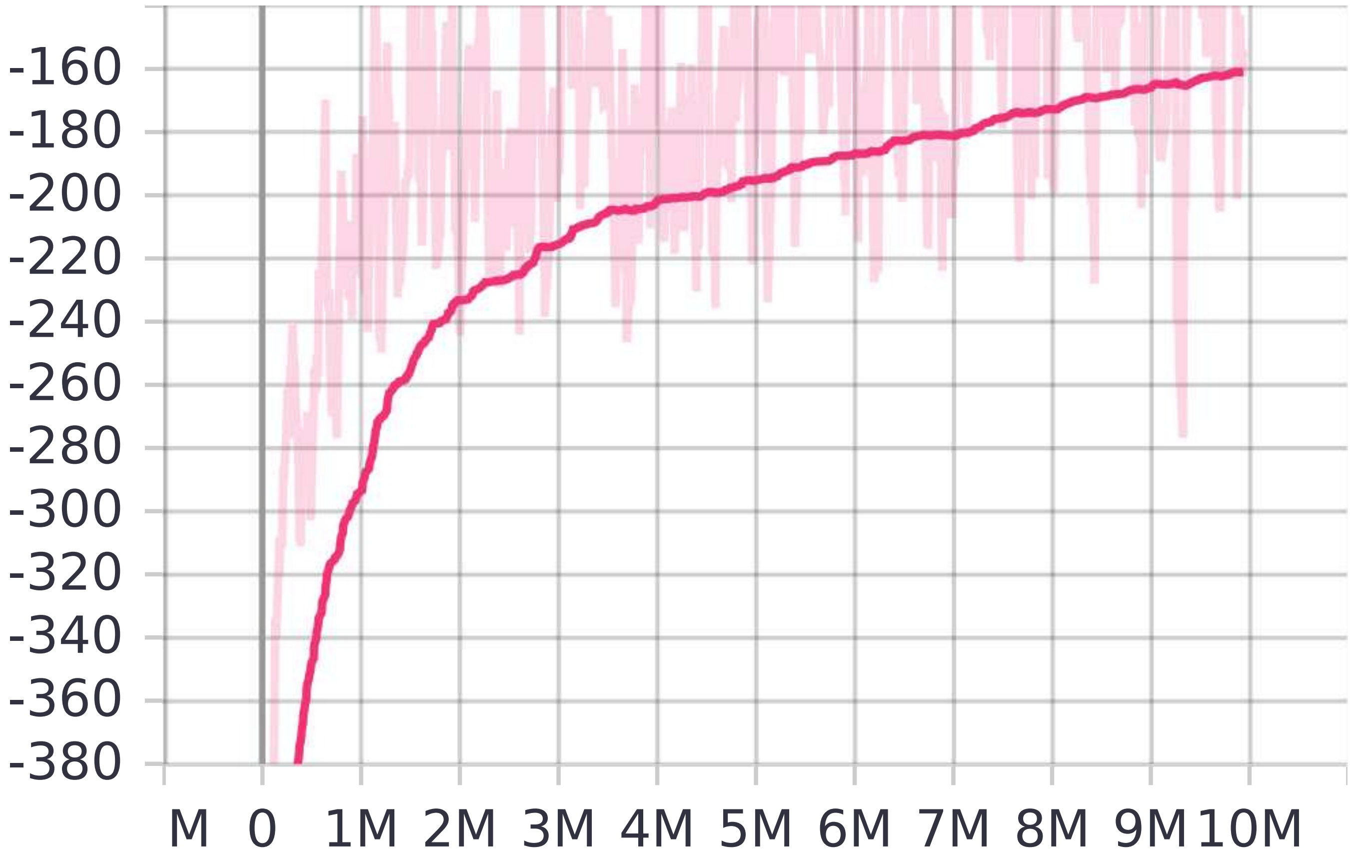


Figure 12. The interaction of the agent with the custom environments

First, for the agent trained for 10 Million steps, the aircraft reached the target 62.3% of the time on average. 69.3% of the time the agent landed on-track (reaching target included), where the agent landed with only an average distance of 0.095 km away from the target position. Furthermore, the average heading error was quite low at 4.65° (See Table 5).

At-Target	62.3 %
On-Track (includes at-target)	69.3%
Distance to target (when on-track)	0.095 km ± 0.086 km
Heading error	4.65° ± 7.63°
Reward	-122.37 ± 393.59

Table 5. Experiment 1: Evaluation of the agent trained in the `Default` environment for 10 Million steps. (All values are averages)

As expected, the agent trained for 15 Million steps, performed mostly better. This can be seen in Table 6. The agent reaches the target successfully with a high value of 73 % of the cases on average and ends on-track (at-target included) even in 81.3% of the cases.

At-Target	73%
On-Track (includes at-target)	81.3%
Distance to target (when on-track)	0.1 km ± 0.1 km
Heading error	12.5° ± 9.4°
Reward	-133.93 ± 369.357

Table 6. Experiment 1 (sub-experiment): Evaluation of the agent trained in the `Default` environment for 15 Million steps. (All values are averages)

A typical successful trajectory of the agents, which were trained in the `Default` environment, can be seen in Figure 13. The heading error is only at around the value of 1.7°.

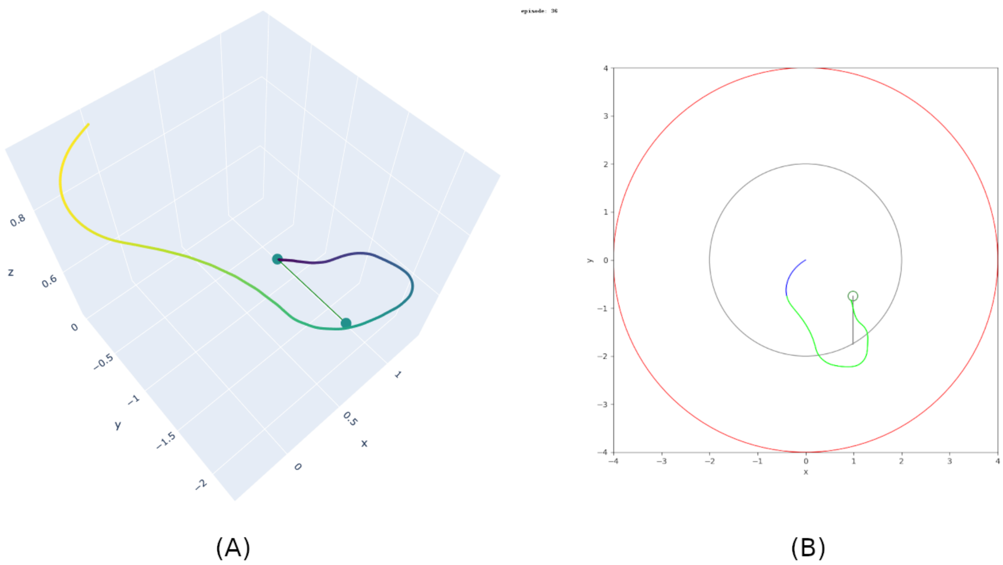


Figure 13. Experiment 1 - Typical trajectory example: The aircraft reaches the target successfully. The average heading error is small. **(A)** 3D view, **(B)** Top view.

7.2 Experiment 2: TwoActions environment

The average rewards during training in the `TwoActions` environment and the `Default` environment were compared. Both were trained for 10 Million steps. Figure 14 shows both reward curves together. As expected, the agent in the `TwoActions` environment seemed to receive much higher rewards earlier during training. Still, surprisingly, the learning in the `TwoActions` environment seemed to have stopped and reached a plateau after 8.5 Million steps. On the other hand, the agent trained in the `Default` environment seemed not to have reached a plateau. (Not even at 15 Million steps, as described in Section 6.1.)

The trained `TwoActions` agent was then evaluated and compared to the agent trained in the `Default` environment. As can be seen in Table 7 the agent trained in the `Default` environment mostly outperforms the `TwoActions` agent. The `TwoActions` agent's average at-target rate reaches only 20.7%, furthermore, its average heading error is around twice as high when compared to the heading error of the `Default` agent, and lastly the average distance to the target, when stopping on-track, is also much higher.

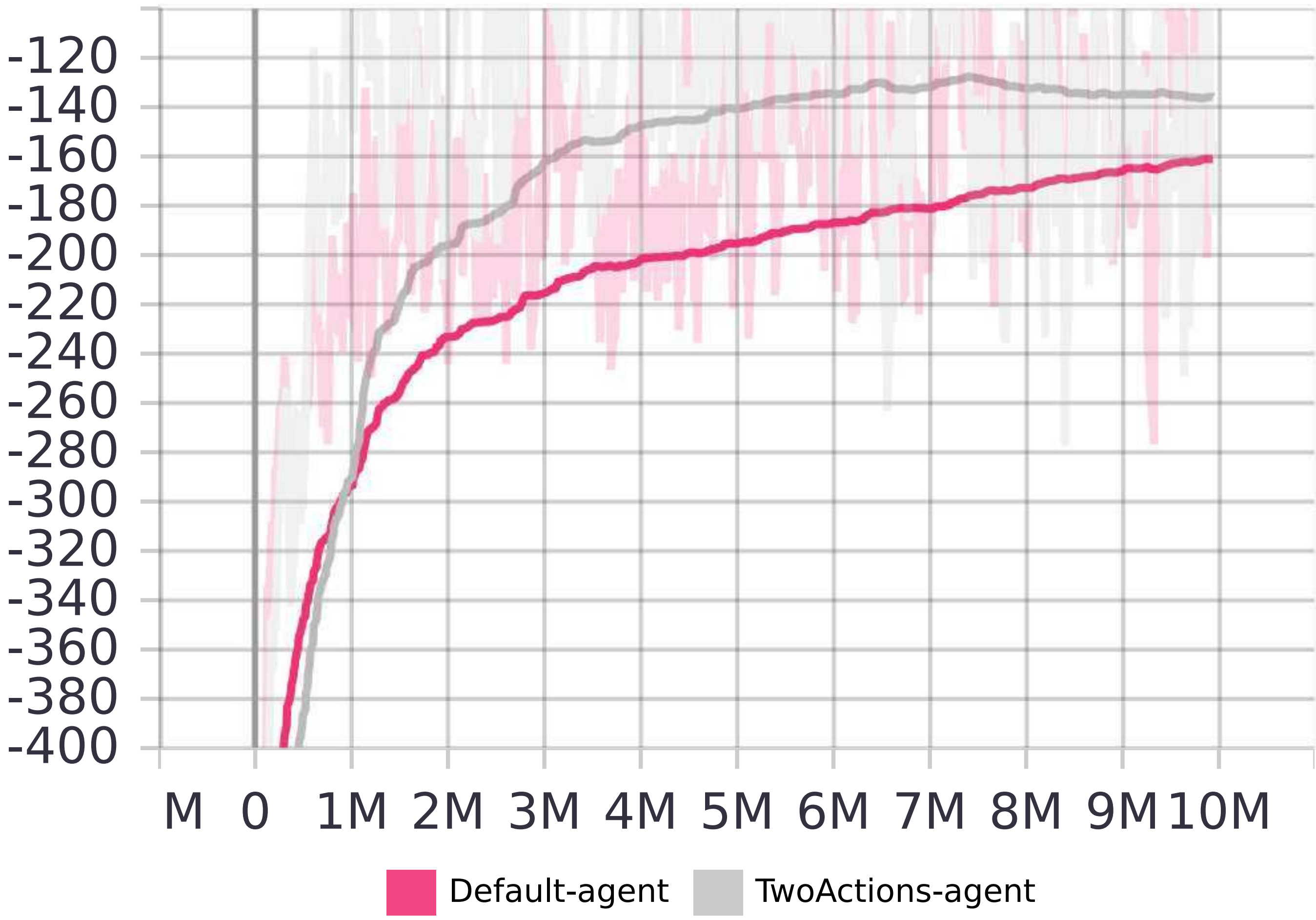


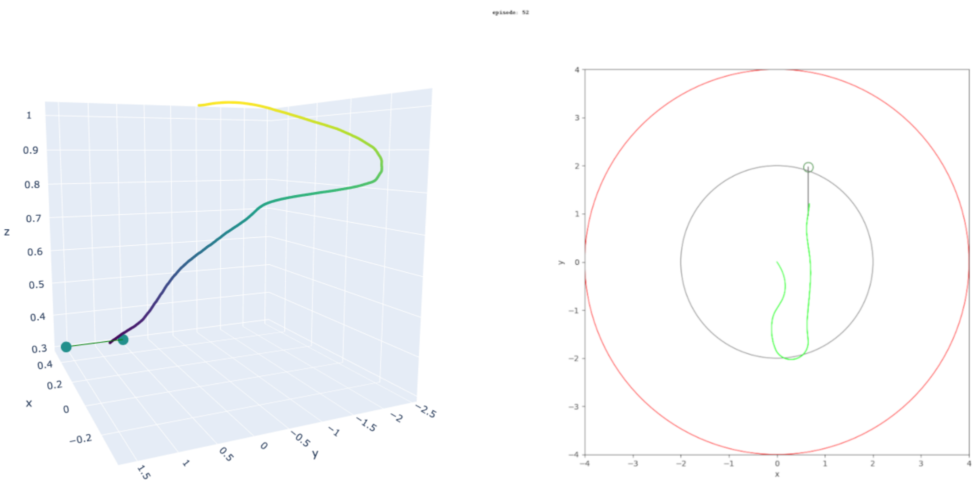
Figure 14. Comparison of the agent trained in the `Default` environment (pink curve) with the agent trained in the `TwoActions` environment (gray curve). The learning of the `TwoActions` agent seemed to have reached a plateau after 8.5 Million steps.

Nonetheless, the `TwoActions` agent achieved a high reward and on-track rate on average, although the on-track rate decreased slightly at around 8.5 Million steps.

	TwoActions	**Default**
At-Target	20.7%	**62.3%**
On-Track (includes at-target)	**77%**	69.3%
Distance to target (when on-track)	0.57 ± 0.45 km	**0.095 km ± 0.086 km**
Heading error	9.43° ± 5.39°	**4.65° ± 7.63°**
Reward	**-61.57 ± 394.56**	-122.37 ± 393.59

Table 7. Evaluation comparison of the `Default` agent and the TwoActions-agent (All values are averages).

In most on-track cases, the agent has led the aircraft first relatively far away from the target and then let the aircraft stay on-track for the rest of the episode, as can be seen in Figure 15.



(A) (B)

Figure 15. Experiment 2 - On-Track: The aircraft lands on-track. The agent seems to have led the aircraft far away first and then kept the aircraft on-track for the rest of the episode.

7.3 Experiment 3: Wind Environment

The evaluation results for the different wind speed ranges are presented in Table 8. As can be seen in the table, with increasing wind speed, the performance of the agent decreased moderately, since the on-track rate and the reward decreased. Moreover, the heading error and the distance to the target (when the aircraft terminates on-track) increased.

Nevertheless, for *all* wind ranges together, on average, the agent reached the target 26.25% of the time. Furthermore, on average, the on-track rate was quite acceptable at 45.25%. This is also true for the heading error at 8.71° and the average value of 0.28 km for the distance to the target.

	0-600 ft/min	**0-1200 ft/min**	**0-2100 ft/min**	**0-3000 ft/min**
At-Target	37.0%	30.3%	22.7%	15%
On-Track (includes at-target)	53.7%	48%	44%	35.3%
Distance to target (if on-track)	0.17 km ± 0.22 km	0.23 km ± 0.28 km	0.30 km ± 0.32 km	0.40 km ± 0.41 km
Heading error	6.98° ± 9.57°	7.75° ± 8.79°	8.0° ± 8.9°	12.21° ± 11.40°
Reward	-49 ± 286	-99.70 ± 330.52	-159.89 ± 374.84	-183.89 ± 370.46

Table 8. Comparison of the wind-agent under different values for the wind speed. The wind is originating from West or East. (All values are averages)

8 DISCUSSION

Three environments were developed to train agents via deep reinforcement learning to achieve the following goal: Guiding an aircraft to an arbitrary target in the 3D space, after a failure of the aircraft's engines. After the training was concluded, the trained agents were evaluated in different experiments.

Overall, in the windless case, it was found that the agent successfully generated trajectories that guided the aircraft to reach the target; or at least terminating very close to it. This was especially true when trained in the `Default` environment. Even when the agent was trained in the `TwoActions` environment, the aircraft stopped mostly on-track and very close to the target. For the agents, trained in the windless environments, the aircraft's heading error at the end of the episode was very low. This was also true for the average distance to the target when ending on-track. These results show that reinforcement learning might have the potential to be a useful tool for flight guidance in an emergency situation.

Experiment 1 showed a relatively high performance after training the agent for around 10 Million steps, and even a superior performance when trained for 15 Million steps. Even at 15 Million steps, no plateau for the average reward was reached. This suggests that there is probably still more room for improvement. For future research, it should be tried to increase the number of training steps.

The findings of experiment 2 were quite surprising. As described in Section 5.4 in detail, the `TwoActions` environment was created to test the following assumption: It was assumed that the agent trained in the `TwoActions` environment should be superior to the agent trained in the `Default` environment. However, the results of experiment 2, unexpectedly, demonstrated the contrary. The agent trained in the `Default` environment (for 10 Million steps) performed better than the `TwoActions` agent. Nevertheless, the `TwoActions` agent reached a higher average reward and high on-track rate after 10 Million steps. When stopping on-track, the distance to the target was relatively low (Table 7), which means that the `TwoActions` agent was capable of successfully leading the aircraft closely to the target position.

To further investigate the reason for this, the generated on-track trajectories were carefully examined, and the following was observed: The `TwoActions` agent first guided the aircraft away from the target and then led it to stay as long as possible on-track, as can be seen in Figure 15 above. This could suggest that the `TwoActions` agent did indeed learn how to receive higher rewards sooner than the agent of the `Default` environment. Interestingly, the `TwoActions` agent probably did this by exploiting the fact that additional rewards were obtained when keeping the aircraft on-track (See Section 5.12). Hence, the `TwoActions` agent could have learned to collect more rewards by staying on-track as long as possible instead of ever reaching the target.

The agent's learned behavior could be a typical example of reward hacking (Ibarz et al. 2018), where the intended behavior, specified by the designer of the reward function, differs from the behavior of the agent (Amodei et al. 2016). Some further research would be needed to confirm this assumption. However, if it is true, then the reward function needs further adjustments. Solving the problem of reward hacking is by itself still an open research question (Clark and Amodei 2016; Amodei et al. 2016). Therefore, for now, there is no simple answer to tackle the issues of reward hacking. Nevertheless, in future research, potential-based reward shaping (Ng, Harada, and Russell 1999) could be applied to adjust the reward function and to probably prevent reward hacking by the agent in this work.

When wind was involved, disappointingly, the agent's performance dropped. An explanation for the drop could be that more training time was necessary, since the `Wind` environment has an elevated difficulty.

Moreover, this indicates, that probably wind-specific dense rewards should be added to the reward function. The added dense rewards could improve the learning process of the agent and reduce training time.

9 CONCLUSION

This work has studied the potential of applying Deep Reinforcement Learning (DRL) to emergency flight guidance after a complete engines-failure of a fixed-wing aircraft. A DRL agent was trained to act as the real-time guidance system in a previously unknown environment. The agent learned to generate a 3D path that led the aircraft from the point of failure to a near, arbitrary-selected landing field.

Three different environments were developed and implemented to train the DRL agent. The integration of curriculum learning and a carefully constructed reward system, which combined sparse and dense rewards, exceedingly improved the agent's learning performance and reduced training times.

Subsequently, the performance of the trained agents were evaluated in the same environments. The results of the evaluation showed the following: In the windless environments, the agents were capable of successfully guiding the aircraft to reach the target in a high number of cases. On the other hand, when wind was involved, the aircraft reached the target successfully in a lower but still significant number of cases. Furthermore, in around half of all cases the aircraft still landed on-track, meaning, mostly in line with the runway center line and with a relatively low heading error.

The results imply that a guidance system based on Deep Reinforcement Learning is potentially a valuable instrument, for guiding an aircraft to a landing field after a complete engines-failure. It could be a significant alternative to systems based on conventional methods. Still, further research is needed, especially in situations where wind is involved. Nonetheless, the results of this work give a first hint of the correct direction.

To our knowledge, no previous work exists that uses DRL to specifically address the presented problem above: Generating a 3D path to a landing field, for a fixed-wing aircraft in case of total loss of thrust. Therefore, the work's presented method and results should contribute to this particular field of study. Furthermore, the developed environments are available as open-source package and can be used for further research [2].

Future research should focus on improving the performance of the agent in occurrence of constant wind. This could be achieved by adapting the reward system and the curriculum learning phases. Moreover, wind coming from different directions and with changing intensities (e.g., wind gradients) should be investigated, to imitate the real-world more precisely.

As described above, the agent was trained using a lightweight aircraft in simulation. Training another agent using the here presented method on other aircraft is relatively easy. Nonetheless, future work could focus on extending the training method to improve the generalization capabilities of one single agent model. By doing so, different aircraft types could be used during deployment, without the need of training and using different agent models for every aircraft.

In this work, the default hyperparameters of the RLlib's TD3 algorithm were mostly left unchanged. Although the used TD3 algorithm requires less parameter hyperparameter tuning than it's predecessor DDPG, future research should study different hyperparameter settings. Automatic hyperparameter search frameworks like Ray Tune (Liaw et al. 2018) or Optuna (Akiba et al. 2019) could be used.

[2] `https://github.com/lauritowal/guidance_flight_env`

Furthermore, a different algorithm, for example Proximal Policy Optimization (PPO) algorithm (Schulman et al. 2017), could be investigated to improve the agent's performance. PPO is another popular state-of-the-art algorithm, that performed exceptional in many other domains.

Lastly, in a real-world flight, static and dynamic obstacles (mountains, other aircraft, etc.) might be present. For this reason, future work should additionally focus on extending the environments to include obstacles.

1 APPENDIX 1

The hyperparameters of the Rllib's TD3 algorithm used for the agent.

```
"twin_q": False,
"policy_delay": 1,
"smooth_target_policy": False,
"target_noise": 0.2,
"target_noise_clip": 0.5,
"evaluation_interval": None,
"evaluation_num_episodes": 10,
"use_state_preprocessor": False,
"actor_hiddens": [400, 300],
"actor_hidden_activation": "relu",
"critic_hiddens": [400, 300],
"critic_hidden_activation": "relu",
"n_step": 1,
"exploration_config": {
    "type": "OrnsteinUhlenbeckNoise",
    "random_timesteps": 1000,
    "ou_base_scale": 0.1,
    "ou_theta": 0.15,
    "ou_sigma": 0.2,
    "initial_scale": 1.0,
    "final_scale": 1.0,
    "scale_timesteps": 10000,
},
"timesteps_per_iteration": 1000,
"evaluation_config": {
    "explore": False
},
"buffer_size": 50000,
"prioritized_replay": True,
"prioritized_replay_alpha": 0.6,
"prioritized_replay_beta": 0.4,
"prioritized_replay_beta_annealing_timesteps": 20000,
"final_prioritized_replay_beta": 0.4,
"prioritized_replay_eps": 1e-6,
```

```
    "compress_observations": False,
    "training_intensity": None,
    "critic_lr": 1e-3,
    "actor_lr": 1e-3,
    "target_network_update_freq": 0,
    "tau": 0.002,
    "use_huber": False,
    "huber_threshold": 1.0,
    "l2_reg": 1e-6,
    "grad_clip": None,
    "learning_starts": 1500,
    "rollout_fragment_length": 1,
    "train_batch_size": 256,
    "num_workers": 0,
    "worker_side_prioritization": False,
    "min_iter_time_s": 1,
```

REFERENCES

(ASA) Federal Aviation Administration (FAA)/Aviation Supplies & Academics (2017) *Pilot's Handbook of Aeronautical Knowledge: FAA-H-8083-25B*. Aviation Supplies & Academics, Incorporated.

Administration, Federal Aviation (2004) "FAA-H-8083-3A" in: *US Department of Transportation Federal Aviation Administration*, pp. 1–33.

Air safety statistics in the EU - Statistics Explained (Aug. 2020) `https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Air_safety_statistics_in_the_EU`. (Accessed on 07/16/2021).

Akiba, Takuya et al. (2019) "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Akkaya, Ilge et al. (2019) "Solving rubik's cube with a robot hand". In: *arXiv preprint arXiv:1910.07113*.

Allerton, David (2009) *Principles of flight simulation*. John Wiley & Sons.

Amodei, Dario et al. (2016) "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565*.

Berndt, Jon (2004) "JSBSim: An open source flight dynamics model in C++". In: *AIAA Modeling and Simulation Technologies Conference and Exhibit*, p. 4923.

Berndt, Jon and Agostino De Marco (2009) "Progress on and usage of the open source flight dynamics model software library, JSBSim". In: *AIAA modeling and simulation technologies conference*, p. 5699.

Berndt, Jon S et al. (2011) "JSBSim reference manual". In: *Work in progress, available at http://jsbsim.sourceforge.net/JSBSimReferenceManual.pdf*.

Berner, Christopher et al. (2019) "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680*.

Best Glide Speed and Distance (2018) `https://www.faa.gov/news/safety_briefing/2018/media/SE_Topic_18-05.pdf`. (Accessed on 07/20/2021).

Bouhamed, O. et al. (2020) "Autonomous UAV Navigation: A DDPG-Based Deep Reinforcement Learning Approach". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* pp. 1–5. DOI: `10.1109/ISCAS45731.2020.9181245`.

Brockman, Greg et al. (2016) *OpenAI Gym*. eprint: `arXiv:1606.01540`.

Clark, Jack and Dario Amodei (Dec. 2016) *Faulty Reward Functions in the Wild*. `https://openai.com/blog/faulty-reward-functions/`. (Accessed on 07/19/2021).

Crocker, David (2007) *Dictionary of aviation: Over 5500 terns clearly defined*. A & C Black.

Dorr, Les (July 2018) *Fact Sheet – General Aviation Safety*. `https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=21274`. (Accessed on 07/16/2021).

Florensa, Carlos et al. (13–15 Nov 2017) “Reverse Curriculum Generation for Reinforcement Learning”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR pp. 482–495. URL: `http://proceedings.mlr.press/v78/florensa17a.html`.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016) *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Gu, Shixiang et al. (2016) “Continuous deep q-learning with model-based acceleration”. In: *International conference on machine learning*. PMLR pp. 2829–2838.

Havenstrøm, Simen Theie, Adil Rasheed, and Omer San (2021) “Deep Reinforcement Learning Controller for 3D Path Following and Collision Avoidance by Autonomous Underwater Vehicles”. In: *Frontiers in Robotics and AI* 7, p. 211.

Henderson, Peter et al. (2018) “Deep reinforcement learning that matters”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32.

Ibarz, Borja et al. (2018) “Reward learning from human preferences and demonstrations in Atari”. In: *arXiv preprint arXiv:1811.06521*.

Klein, Marius, Andreas Klos, Jörg Lenhardt, et al. (2018) “Moving target approach for wind-aware flight path generation”. In: *International Journal of Networking and Computing* 8.2, pp. 351–366.

Klein, Marius, Andreas Klos, and Wolfram Schiffmann (2020) “A Smart Flight Director for Emergency Landings with Dynamical Recalculation of Stable Glide Paths”. In: *AIAA AVIATION 2020 FORUM* p. 3098.

Liaw, Richard et al. (2018) “Tune: A Research Platform for Distributed Model Selection and Training”. In: *arXiv preprint arXiv:1807.05118*.

Lillicrap, Timothy P et al. (2015) “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971*.

Martinsen, Andreas B and Anastasios M Lekkas (2018a) “Curved path following with deep reinforcement learning: Results from three vessel models”. In: *OCEANS 2018 MTS/IEEE Charleston*. IEEE pp. 1–8.

— (2018b) “Straight-path following for underactuated marine vessels using deep reinforcement learning”. In: *IFAC-PapersOnLine* 51.29, pp. 329–334.

Mnih, Volodymyr et al. (2013) “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602*.

Ng, Andrew Y Daishi Harada, and Stuart Russell (1999) “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *Icml*. Vol. 99, pp. 278–287.

Perry, Alexander R (2004) “The flightgear flight simulator”. In: *Proceedings of the USENIX Annual Technical Conference*. Vol. 686.

Raffin, Antonin et al. (2019) *Stable Baselines3*. `https://github.com/DLR-RM/stable-baselines3`.

Randløv, Jette and Preben Alstrøm (1998) “Learning to Drive a Bicycle Using Reinforcement Learning and Shaping.” In: *ICML* vol. 98. Citeseer, pp. 463–471.

Rennie, Gordon (2018) "Autonomous Control of Simulated Fixed Wing Aircraft using Deep Reinforcement Learning". In: `https://purehost.bath.ac.uk/ws/portalfiles/portal/216919613/Rennie_Gordon.pdf`.

Schrittwieser, Julian et al. (2020) "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609.

Schulman, John et al. (2017) "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Silver, David et al. (2017) "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359.

Stephan, Johannes and Walter Fichter (2016) "Fast generation of landing paths for fixed-wing aircraft with thrust failure". In: *AIAA Guidance, Navigation, and Control Conference*, p. 1874.

Sutton, Richard S and Andrew G Barto (2018) *Reinforcement learning: An introduction*. MIT press.

United States Department of Transportation Federal Aviation Administration, Airman Testing Standards Branch (2016) *Airplane Flying Handbook FAA-H-8083-3B*.

Váňa, Petr et al. (2018) "Any-time trajectory planning for safe emergency landing". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE pp. 5691–5696.

Vinyals, Oriol et al. (2019) "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782, pp. 350–354.

Wang, Zhuang et al. (2018) "Design of Agent Training Environment for Aircraft Landing Guidance Based on Deep Reinforcement Learning". In: *2018 11th International Symposium on Computational Intelligence and Design (ISCID)* vol. 2. IEEE pp. 76–79.

Xi, Chenyang and Xinfu Liu (2020) "Unmanned Aerial Vehicle Trajectory Planning via Staged Reinforcement Learning". In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)* IEEE pp. 246–255.

Yan, Chao, Xiaojia Xiang, and Chang Wang (2019) "Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments". In: *Journal of Intelligent & Robotic Systems*, pp. 1–13.

Zhou, Yi et al. (2019) "On the continuity of rotation representations in neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753.