SOSNA

Miasoedov Artemii
Innopolis University
Innopolis, Russia
a.miasoedov@innopolis.university

Brayko Timofey
Innopolis University
Innopolis, Russia
t.brayko@innopolis.university

Artur Rakhmetov
Innopolis University
Innopolis, Russia
a.rakhmetov@innopolis.university

Nikita Kurkulskiu
Innopolis University
Innopolis, Russia
n.kurkulskiu@innopolis.university

I. Introduction

Collision avoidance in transportation refers to the prevention of potential vehicle impacts. This is a broad field, ranging from advanced emergency braking systems in automobiles to space debris avoidance for spacecraft.

Aircraft collision avoidance is among highly relevant topics nowadays. In 2023, NASA's Aviation Safety Reporting System recorded around 300 commercial aircraft near-collisions in the US. The growing number of Unmanned Aerial Vehicles (UAVs) is also a cause of concern.

This paper describes the environment and actor design for a reinforcement learning-based aircraft-aircraft collision avoidance system. However, some techniques might be also used in aircraft-obstacle collision avoidance case studies. We discuss possible implementation strategies, related concepts among with potential pitfalls, demonstrating some of them in practical experiments.

II. Environment

Some researchers have considered 2D environments in studies on similar topics. However, we conducted our experiments in a 3D environment, as this is the domain of the initial problem.

The environment was bounded by a sphere of fixed radius. This was necessary to avoid the issue of agents moving too far from each other during training. The sphere was chosen for several reasons:

1) It is one of the primitive 3D shapes.
2) Relative coordinates inside the sphere always cover the full range in all directions, varying from -1 to 1, for example. This is not true for other primitives, which may affect neural network efficiency.
3) It allows for easy collision and out-of-bounds detection.

The environment didn't include any intentionally placed obstacles, as we focus on aircraft-aircraft collision avoidance.

III. Agents

At least two agents are required to simulate a near-collision case. So, we have identified two possible approaches of near-collision simulation.

A. Cooperative environment

All agents are of the same type and are cooperating to solve the same problem: reach the destination without collisions. This approach has been widely studied by other researchers.

B. Aggressive environment

Agents of two types exist in this scenario. Their goals are opposite, so they are constantly competing with each other:

- Evader — tries to reach destination avoiding any collisions.
- Collider — intercepts the Evader on its way to the destination.

We opted for an aggressive environment. Competition between the agents might lead to more interesting solutions. Meanwhile, the agents in cooperative environment can get stuck in a local optima.

IV. Setup

Unity 3D Engine was the primary tool for our experiments. It has all required features for running physics simulations. Moreover, Unity ML-Agents Toolkit allows training various reinforcement learning agents.

A. Models

A $100\,\mathrm{m}$ sphere represented the environment boundaries.

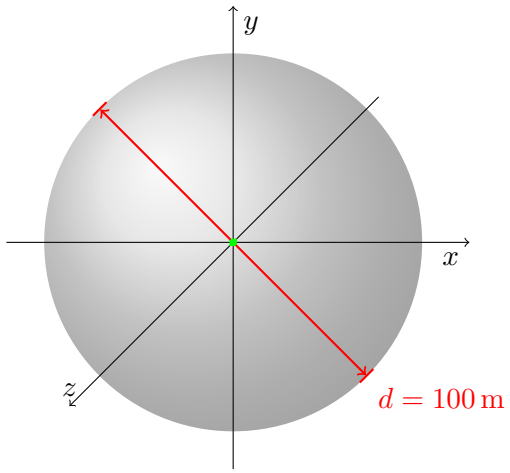

Fig. 1. Environment

A small green target cube with a $1\,\mathrm{m}$ edge was placed in the center of the sphere. Evader agent's goal was to reach this cube. Evader and collider agents were represented as red and blue $1\,\mathrm{m}$ tetrahedrons.

B. Physics

Both agents are equipped with a variable thrust engine. Their controls are similar to an aircraft ones and include yaw, pitch and roll. The gravitation was not considered in our simulations. However, both linear and angular friction forces are enabled. This is done to avoid agent oscillations and destabilization in attempts to stop the movement after some maneuver.

V. Pathfinder

It was our first experience with Unity ML Agents, so we started with a simple agent, pathfinder. Pathfinder would operate in the environment solely, without a collider.

A. Objective

Pathfinder's goal is to reach the target in the center of the sphere from a random position. To simplify the task, we increased the target cube edge from $1\,\mathrm{m}$ to $10\,\mathrm{m}$.

B. Observations

For pathfinder to understand the direction to move, the observations must include the information about the target position and its current position.

Instead of using two separate vectors, we have used a single vector with target position relative to the pathfinder, calculated as:

$$\vec{p} = \vec{p_t} - \vec{p_a}$$

where:

- $\vec{p_t}$ is the target position,
- $\vec{p_a}$ is the agent position.

However, knowing only the relative position to the target is not enough. Pathfinder should understand which direction it is currently moving, so the linear and angular velocities are added to the observations.

Finally, to maneuver, the agent should orient itself in the environment. For example, right turn of agent placed upside down will result in left turn in the environment. So, we have included front, right and up vectors of the agent to the observations. Notably, it is enough to include only a pair of linearly independent vectors to orient the object in 3D space. But we included all three orthogonal vectors, as we thought that it would help the agent to navigate better.

The final observations include:

- Target relative position
- Agent linear velocity
- Agent angular velocity
- Agent front, right, and up vectors

C. Actions

Set of controls:

- Pitch
- Yaw
- Roll
- Throttle

Actions along principal airplane axes are represented as real values from -1 to +1. Positive value corresponds to rotation in clockwise direction, negative – counter-clockwise. Throttle is a real value from 0 to +1.

D. Reward

The reward was simple:

- -0.05 — every time step
- -100 — colliding with the boundaries
- $+10$ — reaching the destination

The agent maximizes its reward, so it was trying to minimize the episode duration. However, ending the episode via self-destruction would lead to a significant penalty. So, environment exploration was the only option for the agent. Distance to the target was not used intentionally in the reward for the sake of interest. The target was relatively big, so even randomly wandering the agent would reach it sooner or later.

E. Neural Network

Neural network consisted of two hidden layers, each with 128 neurons. We trained it to approximate the policy, using the Proximal Policy Optimization, developed by John Schulman. Inputs included a single observation vector. Importantly, all 3D vector values, such as velocity, position, etc., were represented as four real values. Each vector can be represented as a co-directed unit vector multiplied by the magnitude:

$$\vec{v} = \frac{\vec{v}}{|\vec{v}|} |\vec{v}| = \vec{u} |\vec{v}|$$

Such representation may lead to a better neural network performance, as it can help the neural network focus on learning the directional aspects without being biased by the scale of the vectors.

F. Results

The neural network was able to approximate the optimal policy. The agent could successfully navigate from any point in the environment to the target. Moreover, as an experiment, we were manually moving the target during the inference. The agent was able to adjust its trajectory on the fly to reach the destination.

VI. Collider

After the successful training of pathfinder, we switched to a harder task: designing the collider. We decided to use trained pathfinder as the target for collider.

A. Objective

Collider should intercept the pathfinder before it has reached the destination.

B. Observations

During the tests, we figured out that pathfinder is able to reach even moving target. However, we decided to include the target velocities to colliders observations among with pathfinder's observations. So, the final vector would include:

- Target relative position
- Target linear velocity
- Target angular velocity
- Agent linear velocity
- Agent angular velocity
- Agent front, right and up vectors

We didn't include the pathfinder's target in observations, as the collider could potentially learn the pathfinder's behavior.

C. Actions

The set of actions for collider is the same as for pathfinder.

D. Reward

Pathfinder's simple reward didn't work in the collider case, as the chance of randomly hitting the target was significantly lower. That's why the designed reward function had to guide the agent in the right direction. However, the reward should not be extremely strict. Penalizing the agent for not following some strictly defined route would force agent learning that route.

1) Negative distance: The initial idea was to use the distance to the target as a reward function. The agent would be penalized for being far from the target more than getting closer:

$$-|\vec{p_t} - \vec{p_a}|$$

where

- $\vec{p_t}$ is the target position
- $\vec{p_a}$ is the agent position

We could also use inverse distance $|\vec{p_t} - \vec{p_a}|^{-1}$. However, the positive reward from keeping close to the object has to be balanced with negative reward for the episode duration. Otherwise, the agent will circle around the target but not reaching it, increasing its reward infinitely.

Surprisingly, this reward didn't give us reliable results. So, we moved to velocity projection policy.

2) Velocity projection: This reward was designed to hint the correct direction for the agent. Simply projection the linear velocity vector magnitude onto the vector to the target, we will get positive values for heading to the target and negative for going away:

$$\frac{\vec{v_a} \cdot (\vec{p_t} - \vec{p_a})}{|\vec{p_t} - \vec{p_a}|}$$

where

- $\vec{p_t}$ is the target position
- $\vec{p_a}$ is the agent position

Agent trained with this reward was able to follow the direct path to the target. However, the collider was always followed behind the pathfinder, instead of trying to intercept it. This led to the development of a rendezvous distance.

3) Rendezvous distance: The idea behind this reward is to minimize the distance between the closest points in the future. Position along the axis n can be easily

predicted given the velocity function along this axis $v_n(t)$:

$$p_n(T) = \int_0^T v_n(t)dt$$

Given initial position P_n, constant velocity v_n and acceleration a_n, this leads to:

$$p_n(t) = P_n + tv_n + \frac{t^2}{2}a_n$$

So, the distance between two objects x and y in N-dimensional space can be defined as:

$$f(t) = \sum_{n=1}^{N} (p_n^x(t) - p_n^y(t))^2$$

We can find the moment T when the distance is going to be minimal, checking extreme points of this function. To find the extreme points we need to find solutions to:

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = 0$$

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = 2\sum_{n=1}^{N}(\Delta P_n + t\Delta v_n + \frac{t^2}{2}\Delta a_n)(\Delta v_n + t\Delta a_n)$$

where

$$\Delta P_n = P_n^x - P_n^y$$
$$\Delta v_n = v_n^x - v_n^y$$
$$\Delta a_n = a_n^x - a_n^y$$

It can be rewritten in a more compact form:

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = 2(at^3 + bt^2 + ct + d)$$

where

$$a = \frac{1}{2}\Delta\vec{a} \cdot \Delta\vec{a}$$
$$b = \frac{3}{2}\Delta\vec{a} \cdot \Delta\vec{v}$$
$$c = \Delta\vec{P} \cdot \Delta\vec{a} + \Delta\vec{v} \cdot \Delta\vec{v}$$
$$d = \Delta\vec{P} \cdot \Delta\vec{v}$$

Leading to a simple cubic equation:

$$2(at^3 + bt^2 + ct + d) = 0$$

This cubic equation can be solved using modified Cardano's method described in the Numerical Recipes book. Then checking solutions for t we can find the minimal distance in the future. The agent will be penalized using this distance as a negative reward.

The demonstration for these calculations is available in Desmos: https://www.desmos.com/calculator/rgrunpbkbd

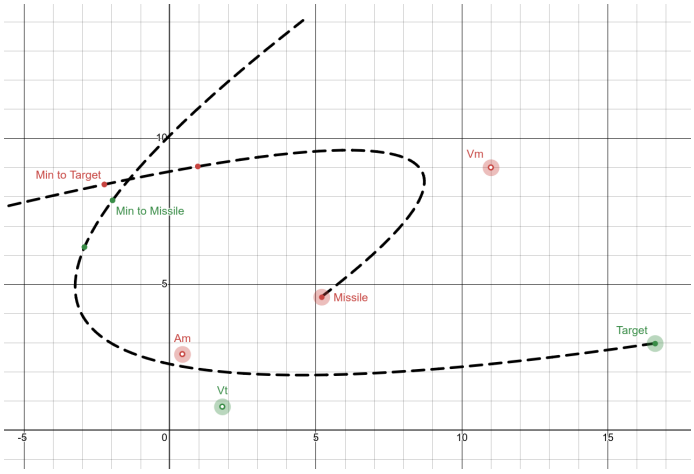

Fig. 2. Prediction of trajectories and closest points

However, even this reward couldn't solve the situation. This was the moment we started to suspect that something was going wrong.

VII. Troubleshooting

After many attempts of training the collider without expected results, we turned back to the pathfinder. This time we didn't increase the size of the target and kept its edge equal to $1\,\mathrm{m}$.

Of course, the initial reward would not work in this case, so we tried the reward functions designed for collider. Surprisingly, none of the rewards worked.

This now seemed as an issue somewhere else. We tried tweaking hyperparameters of PPO, even switching to Soft Actor-Critic (SAC) didn't help. The only thing we hadn't checked until that moment was the observation vector. That was the pitfall...

Vectors we use for the observations reside in the world space. To avoid the disorientation problem, we also passed agent front, right, up vectors to the neural network. As the output, we expected the actions along the aircraft primary control axes. These axes are fixed to the aircraft. So, the actions along these axes are performed in the aircraft space. Therefore, neural network had to figure out the way to transform input vectors from the world space to the aircraft space. Apparently, it couldn't handle this task.

As a solution, we are transforming all input vectors from the world space to the aircraft space. Front, up, and

right vectors were removed from observations, as they became obsolete.

After that we were able to train the pathfinder and collider easily.

VIII. Results

A. Metrics comparison

The collider was successfully trained with both negative and rendezvous distance. Both reward functions performed well in environment with newly pretrained pathfinder.

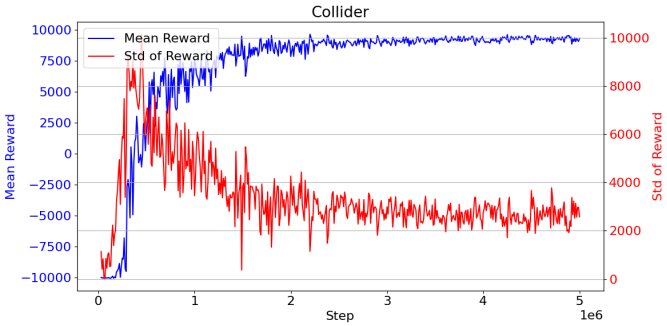

Fig. 3. Mean and std of the collider reward

As it can be seen from 3, the collider reward mean starts from -10,000 (penalty for leaving the bounding sphere). As training progresses, the average reward for the collider increases, while that for the pathfinder decreases 4, indicating that the collider is successfully intercepting it. Finally, collider reward reaches +10,000 (reward for intercepting the pathfinder), while pathfinder reward is getting closer to 0.

Fig. 4. Mean and std of the pathfinder reward

B. Rendezvous vs Negative distance rewards

We were also interested which reward leads to better results: rendezvous or simple negative. To figure it out, we ran 5000 simulations with each agent.

Rendezvous distance shows slightly better results. During the simulations, we saw that agent trained with rendezvous distance reward, easily approaches the pathfinder from different directions, however, bypasses it being extremely close to the target. The reward function might be improved in the future by introducing separate reward for cases when the collider is close to the target.

C. Screenshots

Here are some of the screenshots of our simulations. Blue and red tetrahedrons are collider and pathfinder. Their predicted trajectories are shown as red and blue lines. Green spheres represent the predicted points of minimal distance between the agents.

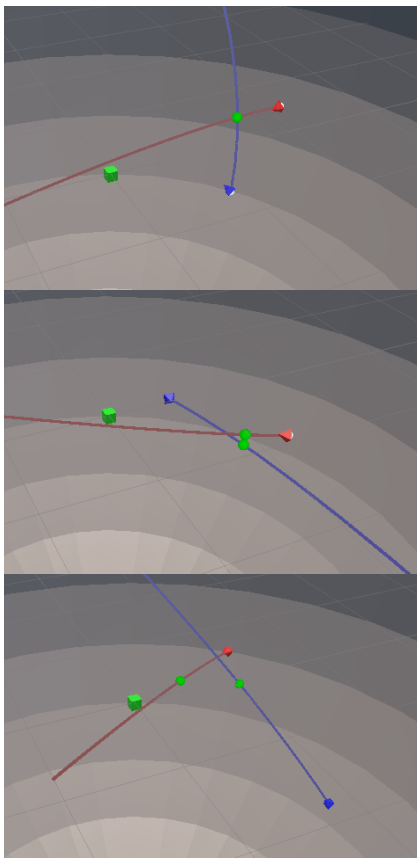

Fig. 5. Trajectories and predicted collisions

IX. Future Work

We were able to train two agents successfully, gaining valuable experience in the environment and reward design. However, we didn't have enough time to train

the evader. The future work would include training the evader. We would like to try hierarchical reinforcement learning approach for the evader. It involves three neural networks:

- Policy selector — determines which policy is used, either Guidance or Evasion
- Guidance controller — guides the agent to the target
- Evasion controller — prevents the collision