Safe Navigation of Indoor Spaces with Deep Reinforcement Learning

Simulation and control of drones for collision-avoidance

Sam Turner, Finlay Sanders

Department of Computer Science
University of Warwick

AI Summit
February 8, 2025

Aims

Problem Identified:

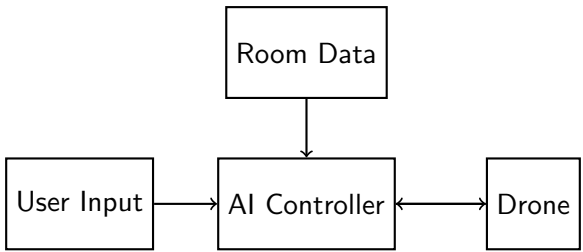
- Drones can be terrifying indoors!
- Especially when controlled by children, or work-in-progress AI models at WAI codenight

Solution:

- Restrict Drone movement to a 'safe-zone'
- By creating an AI powered control layer between the user input and the drone

The Plan

- Extract data about the room using iPhone lidar cameras or structure from motion
- The user will control the drone's target position, and the AI will do the rest
- The AI will be trained using reinforcement learning



Intro to Reinforcement Learning

- Paradigm where an agent with a policy (brain) parametrised by a neural network learns to make decisions by interacting with an environment.
- At each step, the agent observes the current state, takes an action, and receives a reward from the environment.
- Over many steps the agent will learn to maximise its cumulative reward

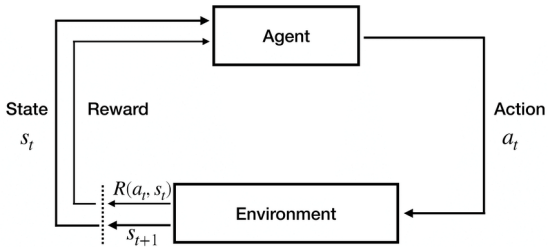


Figure: Sutton, R. S. and Barto, A. G. Introduction to Reinforcement Learning

Simulation Demo

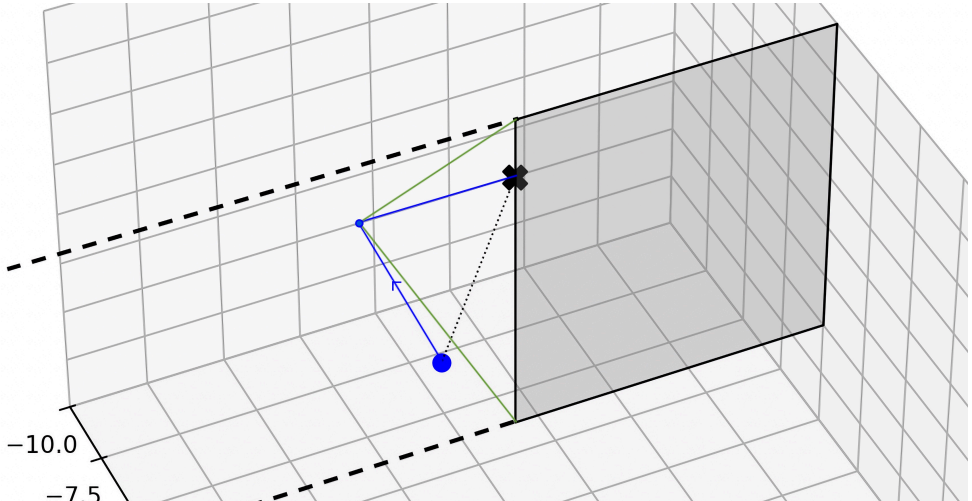
Collision Detection I

- The drone has a collection of six **rays** which it uses to pass collision information directly to the neural network.
- Colliders are modelled as sections of many planes in 3D space.
- To detect the nearest collider, the following process is completed for each plane:
  - Compute the Cartesian equation of the full plane the collider is **co-planar** with.
  - Calculate the point on this full plane which the drone is currently closest to.
  - If this point is already on the collider (check with dot products) then we are done.
  - Otherwise, we need to do some more maths...

$$d(P) = \frac{|ax + by + cz + d|}{\sqrt{a^2 + b^2 + c^2}}$$

Collision Detection II

Calculate the two closest corners of the collider to this point. The closest collision point is on the edge between these corners.



Collision Detection III

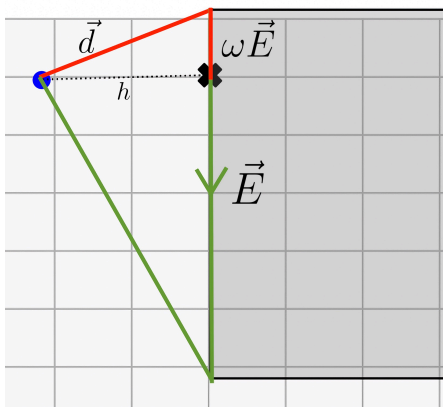
Then calculate the closest point on the collider by finding a value for ω. This can be calculated using a vector projection.

Alternatively, use the vector equation of the line. The closest point is the single solution (shortest distance is always perpendicular) of ω in the following quadratic[a]:

$$h^2 = |\vec{d}|^2 + 2\omega(\vec{E}.\vec{d}) + \omega^2|\vec{E}|^2$$

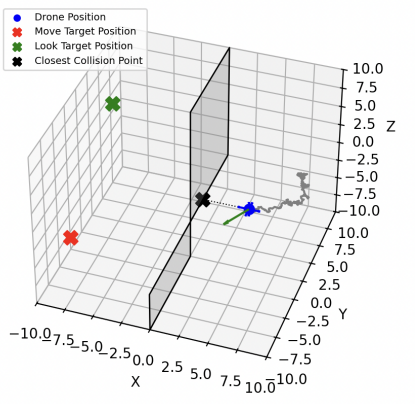
$$\implies \omega = -\frac{\vec{E}.\vec{d}}{|E|^2}$$

[a] If $0 < \omega < 1$, point is along the edge, otherwise it is the corner.



Collision Detection IV

Finally, compute a unit direction vector from the drone to the closest plane. This is projected onto all of the drone's rays with dot products (giving a number between 0 and 1). This represents which direction(s) the collider is in and how far away it is and can be passed straight to the neural network.



Reward Functions I

During reinforcement learning, it is necessary to define a **reward function** which the agent will try to maximise during the training process. We tried many different functions, but settled on one **sparse** reward function and one **dense** reward function, both of which are tested during the training process.

Sparse reward function

$$R = \begin{cases} 1 & \textit{if } \vec{d} \simeq \vec{g}, \\ -1 & \textit{if } C < 0.25, \\ 0 & \textit{otherwise}. \end{cases}$$

Where C is the distance to nearest collider; $\vec{d}, \vec{g}$ are the positions of the drone and goal.

Reward Functions II

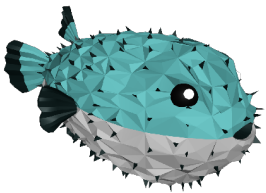
A **sparse** reward function will only reward the agent when it reaches the goal of the current environment, whereas a **dense** reward function will also reward the agent for simply getting closer to the goal. Dense reward functions allow the model to train much faster, but sparse rewards usually result in a more robust model with less noise.

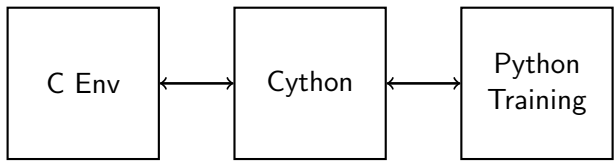
Dense reward function

$$R' = \begin{cases} -0.25 & \textit{if } C < 0, \\ -0.1 + \frac{C}{2} & \textit{else if } C < 0.2, \\ 0.1 & \textit{else if } |\overrightarrow{g_{new}} - \overrightarrow{d_{new}}| > |\overrightarrow{g_{old}} - \overrightarrow{d_{old}}|, \\ 1 & \textit{else if } \vec{d} \simeq \vec{g}, \\ -0.2 & \textit{otherwise}. \end{cases}$$

Training Process and Environment Performance

- Proximal Policy Optimisation (PPO), with a sparse reward scheme.
- Originally wrote the environment in Python + Gymnasium: $\sim$ 2000 SPS.
- Far too slow to run experiments, the model was taking days (30+ hours) to train.
- Re-wrote the environment in C + PufferLib [Suarez, 2024], which we expose to Python using Cython: $\sim 1,000,000$ SPS with 3 fully connected layers of 256 nodes.
- Now, the model can train to a higher standard in less than 30 minutes.





Hyperparameter Sweep

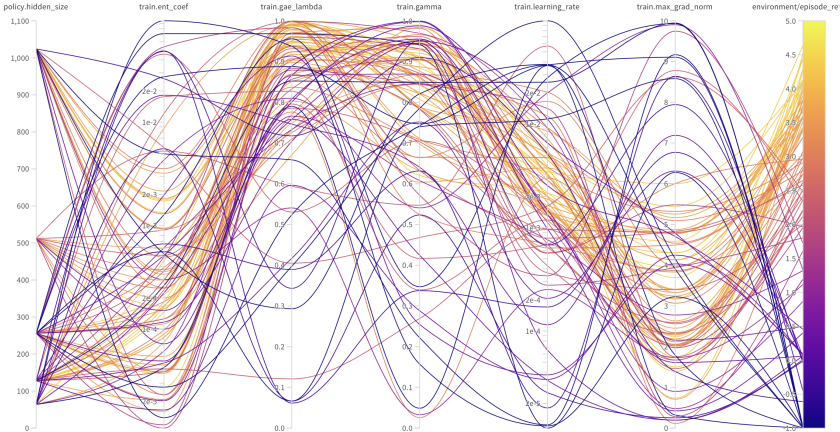


Figure: Hyperparameter sweep with 100 runs of 100 million steps

Room Meshing I

- To control a physical drone, we need to match up the room and the simulation environment
- Essentially finding the 'safe-zone' in a room

Plane Segmentation

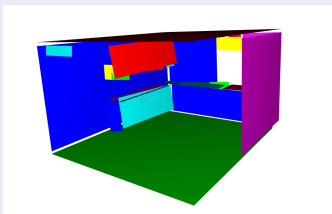


Figure: RANSAC + DBSCAN + convex hull to extract planes

Box Filling

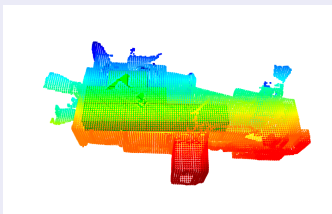
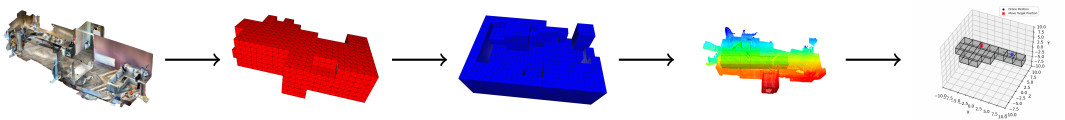


Figure: Fills room point cloud with boxes of a given size outlining the 'safe-zone'

Room Meshing II

- Box Filling turned out to be a lot more simple and reliable



1. Obtain a point cloud of the room using iPhone LiDAR camera (3D Scanner App)
2. Place the point cloud in a 3D grid, and fill in boxes that contain points
3. Flood fill the outside of the shell, then subtract to get the interior (green)
4. Pass interior boxes to the simulation

Results and Future Work

What we built:

- Models that consistently avoided collisions, and were capable of solving pathfinding problems to reach targets
- Trained models that are able to go out of their way to move around walls
- Consistent room segmentation into safe spaces
- A simple interface for controlling the drone and fetching its logs for position estimation

Limitations:

- We struggled to translate model performance into the real world

Future Work:

- Use real time localisation methods like SLAM
- Detect non-static collision objects, such as people

References I


Chen, C., Liu, Y., Kreiss, S., and Alahi, A. (2019).
Crowd-aware robot navigation with attention-based deep reinforcement learning.
In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2255–2262. IEEE.


Chen, Y. F., Liu, M., Everett, M., and How, J. P. (2016).
Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning.
arXiv preprint arXiv:1609.07845.


Everett, M., Chen, Y. F., and How, J. P. (2019).
Collision avoidance in pedestrian-rich environments with deep reinforcement learning.
arXiv preprint arXiv:1910.11689.


Huang, J. et al. (2022).
CleanRL: High-quality single file implementation of deep reinforcement learning algorithms with research-friendly features.

References II

Niu, H., Ji, Z., Arvin, F., Lennox, B., Yin, H., and Carrasco, J. (2021).
Deep reinforcement learning for map-less goal-driven robot navigation.
International Journal of Advanced Robotic Systems, 18(1):1729881421992621.

Rahman, M. M. and Xue, Y. (2022).
Robust policy optimization in deep reinforcement learning.
arXiv preprint arXiv:2212.07536.

Song, S., Saunders, K., Yue, Y., and Liu, J. (2022).
Smooth trajectory collision avoidance through deep reinforcement learning.
arXiv preprint arXiv:2210.06377.

Suarez, J. (2024).
Pufferlib: Making reinforcement learning libraries and environments play nice.

The End

github.com/stmio/drone/