APEX: A Decoupled Memory-based Explorer for Asynchronous Aerial Object Goal Navigation

Supplementary Material

6. Algorithm Flow

Algorithm 1 and Algorithm 2 illustrates the overall workflow of our agent. We employ a multi-process architecture to enable parallel execution of the three modules.

Algorithm 1 Overall Workflow

Input: Observations O
Input: States S
Input: Target Description D

- MEM denotes the shared information, including M_{attr}, M_{expl}, M_{obst}, and others.

function MAP UPDATING(MEM,O,S,D)
  while not IsDetected **do**
    $c, s \leftarrow \text{CAP}(O, D)$
    $\mathcal{M} \leftarrow \text{SEG}(O, c)$
    $MEM \leftarrow \text{Update}(MEM, \mathcal{M}, s, O, S)$
  end while
  process.join()
end function

function ACTION DECISION(MEM,S,O,p)
  while True **do**
    if IsDetected **then**
      Navigate(p)
    else
      break
    end if
    $MEM \leftarrow \text{Update}(MEM, O, S)$
    $a \leftarrow \text{Policy}(MEM, S)$
    Execute(a)
  end while
  process.join()
end function

function TARGET GROUNDING
  while not IsDetected **do**
    $b \leftarrow \text{GD}(O, D)$
    $p \leftarrow \text{RP}(b, O, S)$
  end while
  process.join()
end function

7. More Visualization Results

7.1. Cases

We provide additional visualizations to further illustrate APEX's performance. Fig. 8 shows four navigation episodes: three successful searches for "Car", "Playground", and "Umbrella", and one failure case targeting a "Sandbox". The failure was caused by a collision with a tree, which suggests a potential limitation in our Obstacle Map's ability to detect sparse obstacles.

7.2. Trajectories

Fig. 6 demonstrates the complete trajectories for these four cases (labeled as Case 1-4), in addition to the main paper's example (Case 0).

8. Details of Heuristic Algorithm

The heuristic algorithm mentioned in Ablation Study is detailed in Algorithm 3. The core idea is to simulate each possible action, evaluate the potential reward based on attraction and exploration criteria, and select the action with the highest predicted reward.

9. More Training Details

9.1. Offline Map Generation

Directly incorporating the VLM for attraction map generation within the reinforcement learning loop is infeasible due to its high inference latency. To address this, we pre-generate a complete attraction map for each of the 36 tasks before training begins, which are shown in Fig. 7.

For each task, we designed a flight path to cover the entire relevant search area. An agent was simulated along this path to generate and store a full attraction map. During RL training, instead of calling the VLM, the agent identifies its currently visible grid cells via the depth map and populates its local attraction map by looking up the pre-computed values. This method bypasses the VLM's latency, making the RL training process computationally available.

9.2. Policy Network Architecture

Fig. 5 illustrates the architecture of the policy network in our RL-based Action Decision Module. It utilizes three CNNs to extract features from the Attraction, Exploration, and Obstacle maps, respectively. To process the 3D map data, the z-axis is treated as the channel dimension for the

Algorithm 2 Memory Updating Workflow

Input: Masks $\mathcal{M}$
Input: Observations O
Input: States S
Input: Attraction Scores s

function ATTRUPDATE($M_{\text{attr}}, s, \mathcal{M}, O, S$)
  $\mathcal{G}_{\text{vis}}$: the set of visible grid cells in the current view.
  for $g \in \mathcal{G}_{\text{vis}}$ **do**
    P_g: the set of pixels from O that project to g.
    $Mask^* \leftarrow \arg\max_{Mask_i \in \mathcal{M}} |P_g \cap Mask_i|$
    if $Mask^*$ is found **then**
      $s^* \leftarrow$ score in s corresponding to $Mask^*$
      $d_{\text{new}} \leftarrow \text{Dis}(S.\text{pos}, g.\text{center})$
      if $d_{\text{new}} < M_{\text{attr}}[g].\text{dis}$ **then**
        $M_{\text{attr}}[g].\text{score} \leftarrow s^*$
        $M_{\text{attr}}[g].\text{dis} \leftarrow d_{\text{new}}$
      end if
    end if
  end for
  return M_{attr}
end function

function EXPLUPDATE(M_{expl}, O, S)
  $\mathcal{P}_{\text{end}}$: the set of 3D world points from O.
  $\mathcal{G}_{\text{traversed}} \leftarrow \text{TraceRay}(S.\text{pos}, \mathcal{P}_{\text{end}})$
  for $g \in \mathcal{G}_{\text{traversed}}$ **do**
    $d \leftarrow \text{distance}(S.\text{pos}, g.\text{center})$
    $v_{\text{current}} \leftarrow M_{\text{expl}}[g]$
    $v_{\text{new}} \leftarrow \text{ExplRate}(d, v_{\text{current}})$
    $M_{\text{expl}}[g] \leftarrow v_{\text{current}} + v_{\text{new}}$
  end for
  return M_{expl}
end function

function OBSTUPDATE(M_{obst}, O, S)
  $\mathcal{G}_{\text{depth}}$: the set of grid cells hit by depth rays from O.
  $M_{\text{obst}}[g] \leftarrow 1, \quad \forall g \in \mathcal{G}_{\text{depth}}$
  return M_{obst}
end function

function TRACERAY($S_{\text{start}}, \mathcal{P}_{\text{end}}$)
  $\mathcal{G}_{\text{traversed}}$: the set of grid cells
  for $p_{\text{end}} \in \mathcal{P}_{\text{end}}$ **do**
    $\vec{r} = p_{\text{end}} - S_{\text{start}}$
    Sample points $\mathcal{P}_s$ along $\vec{r}$ at regular intervals.
    $\mathcal{G}_{\text{traversed}} \leftarrow \text{RP}(\mathcal{P}_s)$
  end for
  return $\mathcal{G}_{\text{traversed}}$
end function

Algorithm 3 Heuristic Action Decision

Input: Attraction Map M_{attr}
Input: Exploration Map M_{expl}
Input: Obstacle Map M_{obst}
Input: Current State s

- Let a be an action from the action set $\mathcal{A}$.
- Let s' be the hypothetical state after action a.
- The **attraction reward** R_{attr} and **exploration reward** R_{expl} is defined in main paper.

Initialize R to store the predicted reward.
for $a \in \mathcal{A}$ **do**
  $s' \leftarrow \text{Simulate}(a, s, M_{\text{attr}}, M_{\text{expl}}, M_{\text{obst}})$
  if IsUnsafe(s') **then**
    $R[a] \leftarrow -\infty$
    continue
  else
    $R_[a] \leftarrow W_{\text{attr}} R_{\text{attr}}(s') + W_{\text{expl}} R_{\text{expl}}(s')$
  end if
end for
return $\arg\max_{a \in A} R[a]$

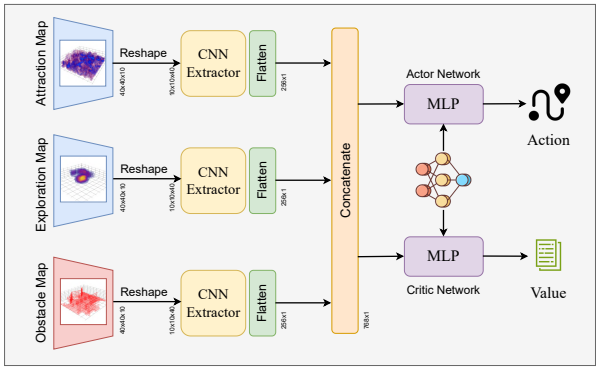

Figure 5. The architecture of policy network.

convolutional layers. The resulting feature vectors are concatenated and shared between an Actor (MLP) and a Critic (MLP) head. The Actor network outputs the final action policy. This implementation is based on the MultiInputPolicy architecture provided by the Stable-Baselines3 [29].

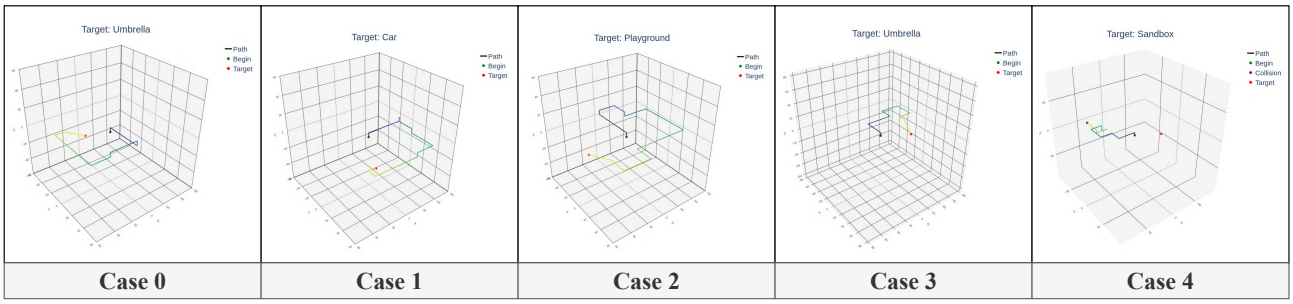

Figure 6. Complete trajectories of visualization examples.

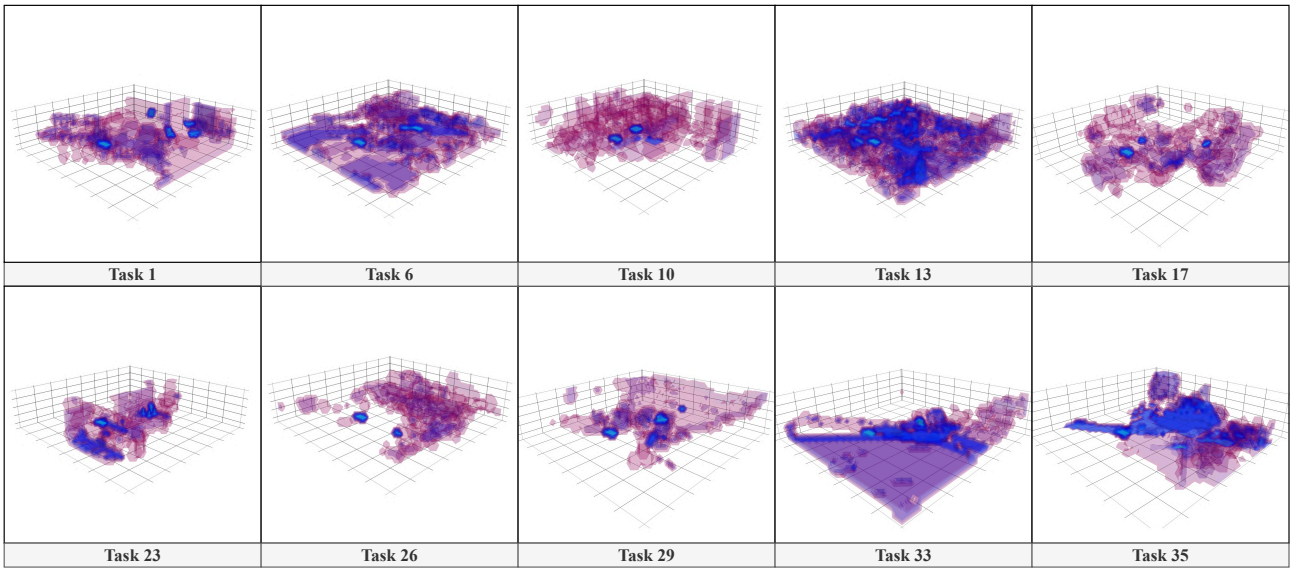

Figure 7. Examples of collected attraction maps.

Figure 8. More visualized results. The first three examples are successful cases, with the targets ”Car”, ”Playground”, and ”Umbrella” respectively. The last one is a failure case, with the target being ”Sandbox”.