Active Neural SLAM

Tuesday, 20 October 2020 11:34

- **Active Neural SLAM (ANS)**
  a modular and hierarchical approach to learn policies for exploring 3D environments.
- Uses analytical path planners with learned ① SLAM module ② global ③ local policies.
- In the SLAM module: learning provides flexibility w.r.t. input modalities.
- In global policies: learning leverages structural regularities of the world.
- In local policies: learning provides robustness to errors in state estimation.
- Navigation problems:
  1. point goal tasks: navigating to specific coordinates.
  2. semantic navigation: finding the path to a specific scene or object.
  - A core problem for navigation in unknown environments is exploration.

  Learning exploration policies for navigation [Chen'19];
  used end-to-end learning
  pros: 1. flexibility w.r.t. input modalities (use RGB images)
  2. robustness to errors in state estimation.
  3. leverage structural regularities of the real world
  → more efficient behavior in previously unseen environment,
  cons: expensive path-planning (uses imitation learning).
- Solution:
  learning (in the 3 pros) happens at different time scales
  → can be factored out.
- Learning-based navigation techniques: NN policies that reason via spatial representations
  Neural Map: structured memory for DRL [Parisotto'18]
  Neural SLAM [Zhang'17]
  ① topological representations
  Episodic curiosity through reachability [Savinov'19]
  ② differential and trainable planners
  Gated path planning networks [Lee'18]
- Task setup:
  the objective: maximizing the coverage in a fixed time budget.
  the coverage: the total area in the map known to be traversable.
  Habitat simulator [Savva'19]
  - Actuation and sensor noise model:
    agent pose (x, y, o), control command $\Delta u_a = (x_a, y_a, o_a)$
    - The actuation noise ϵ_{act}: is the difference between the actual agent pose p_t after the action and the intended agent pose $(p_t + \Delta u)$
      $\epsilon_{act} = p_t - (p_t + \Delta u) = (x^* - x_a, y^* - y_a, o^* - o_a)$
    - The sensor noise ϵ_{sen}: is the difference between the sensor pose estimation p_t' and the actual pose p_t
      $\epsilon_{sen} = p_t' - p_t = (x' - x^*, y' - y^*, o' - o^*)$
  - Actions (control commands):
    Forward: $u_F = (0.25, 0, 0)$
    Turn right: $u_R = (0, 0, -10^\circ\pi/180)$
    Turn left: $u_L = (0, 0, 10^\circ\pi/180)$
  - For each action, a seperate Gaussian Mixture Model GMM is fitted for the actuation and sensor noise, making the total 6 models.
- **Methods:**

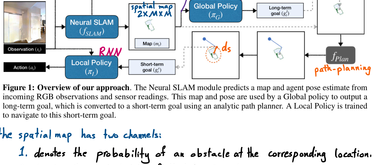

Figure 1: Overview of our approach. The Neural SLAM module predicts a map and agent pose estimate from incoming RGB observations and sensor readings. This map and pose are used by a Global policy to output a long-term goal, which is converted to a short-term goal using an analytic path planner. A Local Policy is trained to navigate to this short-term goal.

- the spatial map has two channels:
  1. denotes the probability of an obstacle at the corresponding location.
  2. denotes the probability of that location being explored.
     (explored = known to be free space or an obstacle.
- **Neural SLAM module:**

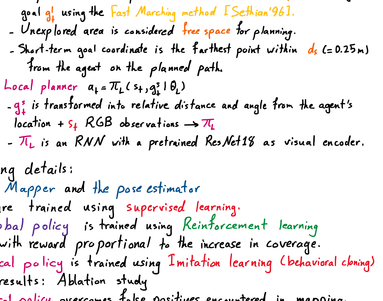

Figure 2: Architecture of the Neural SLAM module: The Neural SLAM module (f_{SLAM}) takes in the current RGB observation, o_t, the current and last sensor reading of the agent pose $z_{t-1:t}$, last agent pose estimate, $\hat{x}_{t-1}$ and the map at the previous time step m_{t-1} and outputs an updated map, m_t and the current agent pose estimate, $\hat{x}_t$. 'ST' denotes spatial transformation.

$$m_t, \hat{x}_t = f_{SLAM}(s_t, \hat{x}_{t-1:t}, \hat{x}_{t-1}, m_{t-1} | \theta_S)$$

- the trainable parameters θ_S consists of:
  1. the mapper f_{map}:
     outputs a egocentric top-down 2D spatial map $p_t^{ego} \in [0,1]^{2 \times V \times V}$ (V is the vision range)
     predicts the ① obstacles and the explored area in the current observation.
  2. the pose estimator f_{PE}:
     predicts the agent pose $\hat{x}_t$ based on ① past pose estimate $\hat{x}_{t-1}$ and ② the last two egocentric map predictions $p_{t-1:t}^{ego}$
     - predicts the pose change by comparing maps after transforming the last to the current frame.
- Global policy $g_t^* = \pi_G(h_t | \theta_G)$
  Its input $h_t \in [0,1]^{4 \times M \times M}$ has 4 channels:
  1-2. the spatial map m_t given by SLAM.
  3. current position estimation $\hat{x}_t$ given by SLAM.
  4. visited locations $\hat{x}_t^k : k \in \{0, \ldots, t\}$
  two transformations performed on h_t:
  1. subsample a window of size 4XGXG around the agent from h_t.
  2. perform max pooling operations to get an output size 4XGXG.
  → the input to the Global policy is a tensor 8XGXG
  - the output g_t^* is a long-term goal in GXG space;
    π_G is a convolutional neural network.
- Planner $g_{ts} = f_{plan}(g_t^*, m_t, \hat{x}_t)$
  - computes the short-term goal g_{ts}^*;
  - computes the shortest path from the current location $\hat{x}_t$ to the long-term goal g_t^* using the Fast Marching method [Sethian'96].
  - Unexplored area is considered free space for planning.
  - Short-term goal coordinate is the farthest point within d_t (= 0.25m) from the agent on the planned path.
- Local planner $a_t = \pi_L(s_t, g_{ts}^* | \theta_L)$
  - g_{ts}^* is transformed into relative distance and angle from the agent's location + s_t RGB observations → π_L
  - π_L is an RNN with a pretrained ResNet18 as visual encoder.
- **Training details:**
  - The Mapper and the pose estimator are trained using supervised learning.
  - Global policy is trained using Reinforcement learning with reward proportional to the increase in coverage.
  - Local policy is trained using Imitation learning (behavioral cloning)
- From results: Ablation study
  - Local policy overcomes false positives encountered in mapping.
  - Global policy ignores small spaces and choose distant long-term goals → higher coverage.
  - ANS reache performance 0.789/0.703 SPL / Succ in 1 million frames
    Succ: success rate.
    SPL: success weighted by path length (normalized inverse).
    ANS > FB X speed-up as compared to best RL baseline.
- **Architecture:**
  f_{map}: ResNet 18 convolutional layers → embedding
  → 2 fully connected layers
  → 3 deconvolutional layers → first-person top-down 2D map
  binary cross-entropy loss X1
  f_{PE}: 3 convolutional layers
  → 3 fully connected layers
  MSE loss X1000
  π_G: PPO with reward = increase of coverage m^2 X 0.02
  π_L: binary cross-entropy loss.