Abstract
Humans can move their bodies and eyes actively to perceive the state of the environment they are surrounded by. Autonomous robots are designed to perceive the state of the environment they are surrounded by. Autonomous robots are needed to learn this ability, so called active perception, to behave as humans do. In this paper, we propose a reinforcement learning algorithm to make the robots have the perceptual ability. In our algorithm, we simultaneously train two agents which control the robot and its sensor on the robot to achieve a task. We conducted experiments on navigation tasks in a 3D environment where useful information for the task achievement is partially occluded. The experimental results show that our algorithm can obtain better perceptual behavior and achieve a task with a higher success rate than conventional reinforcement learning algorithms.

Keywords
Active Perception; Reinforcement Learning; Autonomous Robot

1 Introduction
While autonomous robots are designed to perceive the state of the environment via various sensors, their decision making is often hindered due to sensor occlusions. Active perception is used to enable autonomous robots to actively perceive the necessary information for decision making in such a situation [1, 7]. There are some deep reinforcement learning (RL) studies that are related to active perception, however, their applicability is limited just in scanning or mapping [4, 5].

In this paper, we simultaneously train two agents which make different roles, one is a moving agent to behave for achieving a given task while moving the robot, the other is a perceptual agent to actively explore the state of the environment for better task achievement. For learning the perceptual policy, we introduce a meta-evaluation process which measures how much the perceptual policy improves the behavior of the moving policy. We conducted experiments on navigation tasks in a 3D simulation environment with some obstacles, and the results show that our algorithm can obtain better perceptual behavior and achieve a task with a higher success rate than conventional RL algorithms.

2 Problem Description
Let \( a^m \) denote the moving agent that controls the robot’s movement and \( a^p \) denote the perceptual agent that decides where to face the sensor, which is assumed to be a camera mounted on the robot in this paper. We propose a new Markov decision process (MDP) \( M = (S, U^m, U^c, \pi^m, \pi^c, r, p_1, \gamma) \) in our problem setup, where \( S \) is a set of states, \( U^m \) and \( U^c \) are sets of moving and perceptual agent’s actions. \( \pi^m : S \times U^m \times S \to [0, 1] \) and \( \pi^c : S \times U^c \times S \to [0, 1] \) are the state transition functions, \( r : S \times U^m \to \mathbb{R} \) is a reward function, \( p_1 : S \to [0, 1] \) is a distribution over initial states, and \( \gamma \in (0, 1) \) is a discount factor, respectively. We define a stochastic policy \( \pi \) as the probability distribution over actions conditioned on the current state: \( \pi : S \times U \to [0, 1] \). For a given \( \pi \), the state value is defined as \( V^\pi(s) = \mathbb{E}_{\tau} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, u_t) \right] \), where \( s_0 = s \) and \( u_0 \sim \pi(\cdot | s_0) \). We assume a realistic setting where the robot’s movements affect which direction the camera faces whereas the camera’s movements do not affect the robot’s movements. The state transition in this setting is described as follows:

1. Given a state \( s \in S \), the moving agent \( a^m \) takes an action \( u^m \in U^m \) according to the moving policy \( \pi^m : S \times U^m \to [0, 1] \).
2. The perceptual agent \( a^p \) receives a state \( s' \in S \) induced by the state transition function \( \pi^m(\cdot | s, u^m) \), and takes an action \( u^c \in U^c \) given the state \( s' \) according to the perceptual policy \( \pi^c : S \times U^c \to [0, 1] \).
3. The moving agent \( a^m \) receives the next state \( s'' \) induced by the other state transition function \( \pi^c(\cdot | s', u^c) \).

Based on the transition, the joint distribution \( p^{m,c}_{\pi^m,\pi^c} \) of a state trajectory \( s_{\tau} = \{s_1, s_1, \ldots, s_{\tau-1}, s_{\tau}\} \) induced by rollouts following \( \pi^m, \pi^c \),
\(\pi^c, P^m, \text{and } \mathcal{P}^c\) is thus written as follows:

\[
p_{m,c}(s_t) = p_1(s_1) \prod_{t=1}^{T} \mathcal{P}^c(s_{t+1}|s_t, u^c_t)\pi^c(u^c_t|s_t) \\
= p_1(s_1) \prod_{t=1}^{T} \mathcal{P}(s_{t+1}|s_t, u^m_t)\pi^m(u^m_t|s_t),
\]

where \(\mathcal{P}(s_{t+1}|s_t, u^m_t) = \mathcal{P}^c(s_{t+1}|s_t, u^c_t)\pi^c(u^c_t|s_t)\mathcal{P}^m(s_t|s_t, u^m_t)\) is a state transition function that involves \(\pi^c\).

### 3 Optimization of Policies

In our setting, we prepare two same environments : \(E_1\) and \(E_2\). In the environment \(E_1\), the moving policy \(\pi^m_\theta\) runs with a perceptual policy \(\pi^c_\phi_1\), while the value function approximator \(V_{w_1}\) estimates the expected cumulative rewards the moving agent \(a^m\) earns. In the environment \(E_2\), another perceptual policy \(\pi^c_\phi_2\) and the value function approximator \(V_{w_2}\) make the same role of \(\pi^c_\phi_1\) and \(V_{w_1}\), respectively. The policies \(\pi^m_\theta\), \(\pi^c_\phi_1\), and \(\pi^c_\phi_2\) are parameterized by \(\theta, \phi_1\) and \(\phi_2\), the value function approximators \(V_{w_1}\) and \(V_{w_2}\) are parameterized by \(w_1\) and \(w_2\), respectively. We define a meta-evaluator \(\mathcal{V}\) which measures how much perceptual policy \(\pi^c_\phi\) is better suited for the moving agent \(a^m\) than \(\pi^c_\phi\) by calculating the difference of the average values \(V_{w_2}\) and \(V_{w_1}\) given the trajectories in each environment. The average value \(V_{w_1}\), which is calculated by the value function approximator \(V_{w_1}\) updated in the environment \(E_1\), summarizes how much cumulative rewards the moving policy \(\pi^m_\theta\) can obtain with a perceptual policy \(\pi^c_\phi\) in the environment \(E_1\) for \(\forall \epsilon \in \{1, 2\}\). The meta-evaluator \(\mathcal{V}\) can be calculated as follows:

\[
\mathcal{V}(\tau_k) = \mathcal{V}_{w_1}(\tau_k) - \frac{1}{K} \sum_{k=1}^{K} \mathcal{V}(\tau_k),
\]

where \(\tau_k = \{s_t, u^c_t, r_t, \tau_k, u^c_t\}_{t=1}^{T}\) is a trajectory for \(\forall \kappa \in \{1, 2, \cdots, K\}\) and \(K\) is the number of trajectories. \(D_1 = \{\tau_k\}_{k=1}^{K}\) denotes a set of trajectories obtained by rollouts of the policies in the environment \(E_1\). We update \(\pi^c_\phi\) using the framework of REINFORCE [8] by replacing the cumulative rewards with the meta-evaluator \(\mathcal{V}\). To update the \(\pi^c_\phi\) to approximate what \(\pi^c_{\phi_1}\) is used to be, we employ the soft update rule as follows:

\[
\phi_1 = \alpha \phi_2 + (1 - \alpha) \phi_1
\]

where \(\alpha\) controls rate of the updates and we set \(\alpha = 0.05\). After the soft update, we employ the general actor-critic algorithm to update both the moving policy \(\pi^m_\theta\) and the value function approximator \(V_{w_1}\) by using \(D_1\) sampled again by a rollout of \(\pi^m_\theta\) with \(\pi^c_{\phi_1}\). Since the perceptual policy \(\pi^c_{\phi_1}\) is expected to be better suited for the moving policy \(\pi^m_\theta\) than the one before update in (3), the moving policy \(\pi^m_\theta\) is updated so that \(\pi^m_\theta\) gets further more rewards with the better perceptual policy \(\pi^c_{\phi_1}\). The process of updating policies is completed by copying \(w_1\) to \(w_2\) for the next update. In the training process, we empirically found that the \(\epsilon\)-greedy strategy is beneficial to learn better perceptual policies. We sample the action \(u^c_\epsilon\) of the perceptual agent in the environment \(E_2\) as follows:

\[
u^c_\epsilon(t) = \left\{ \begin{array}{ll} \arg \max \pi^c_{\phi_1}(s_t) & \text{with probability } 1 - \epsilon \\ \text{a random action} & \text{with probability } \epsilon \end{array} \right.
\]

After training, we employ \(\pi^c_{\phi_2}\) as the actual perceptual policy with which the moving policy \(\pi^m_\theta\) performs at the test time.

### 4 Experiments and Results

We build the environment by using the Mini-world package[3]. It has a single room where multiple boxes (green, yellow, blue) are placed between the agent and the goal (red box) as obstacles. We perform navigation tasks and evaluate our algorithm (\(\epsilon = 0.1, 0.3\)) against the following four baselines: **Fixed_Foreward** (only the moving policy is trained with the camera is fixed in the forward direction), **Joint** (a single agent with the joint action space \(\mathcal{U}^m \times \mathcal{U}^c\)), **IA2C** (\(a^m\) as well as \(a^c\) train independently using the cumulative reward), **Curriculum** (inspired by [2], after the **Joint** agent is pre-trained in the environment without any obstacles, its parameters are used as initial values for training in the environment with the obstacles). The moving agent \(a^m\) for all methods are trained by advantage actor-critic (A2C) [6].

Figure 1 shows the experimental results with two horizontal fields of view 60 (Figure 1(d)) and 100 (Figure 1(e)) degrees during training. In the setting with 100 degrees, some baselines obtain large returns. Even in this setting, our algorithm with \(\epsilon = 0.1\) yields the best result. On the other hand, although all methods deteriorate the average returns in the setting with 60 degrees, our algorithm with \(\epsilon = 0.3\) achieves the highest success rate. These results show that our algorithm makes autonomous robots learn perceptual behaviors for better task achievement, and are consistent with our intuition – the less information we have, the more we are going to look at our surroundings.
REFERENCES