Review: Sim-to-Real Transfer of Wall Following Model

1st Ian Sinclair
Ritchie School of Engineering and Computer Science
University of Denver
ian.sinclair@du.edu

***Abstract*—The following is a review of the transfer of a reinforcement learning model trained in simulation to a real-world robot. Including a brief description of the model, the desired behavior, and its success in simulation. Along with challenges and special considerations that were taken while adapting the model to the real world. This highlights a key problem in robotic research in AI, or rather as a consequence of time and/or resource constraints, complex models are typically trained in simulated environments that are not fully representative of the real world. And so what considerations must be taken to ensure the model is robust in the real world? Within this report, we cover a few algorithmic decisions made during the real-to-sim transfer that may lead to a better result. Including, removing artificial bottlenecks like .sleep commands and improving the simulation with domain randomization.**

I. Introduction

Complex robotic behavior is difficult to actualize using traditional control schemas, especially in a dynamic environment. This a large contributor to the success of Reinforcement learning (RL) models in robotics, which is able to learn behavior without the need for involved kinematic models that may be intractable. Generally, RL admits a function, π^* that maps a set of states to a set of actions, $\pi^* : S \rightarrow A$, which is learned through a large sequence of trial-and-error attempts. During training, the model will assume poor function, π, then, given the robot is in state $s \in S$, will take action $a \in A$ and observe/ update π from the result. In this way, after each decision, good or bad, the behavior of the robot is improved. However, this may take many-many trials and comes with the potential to damage the robot in the event of testing poor decisions. And so simulated environments that can run much faster than real-time becomes pivotal. Unfortunately, in order to decrease computation complexity, these environments are often simplified when compared to the real world. And so there remains a question of how to convert a model from simulation to the real world while maintaining robustness. The remainder of this review covers the wall-following problem on a high level, the RL model implementation including state and action spaces. Along with special conditions that were changed to support the sim-to-real transfer.

II. Wall-Following Problem Statement

Our task is to train a robotic agent to travel the circumference of its environment without crashing into or losing walls. In general, this means that if the robot is a some desired distance, d_w from a wall at time t_i, then this distance is maintained at time t_{i+1}. In particular, we are examining four behaviors, being 1) following a straight wall, 2) making a 90° turn, 3) making a 180° turn around a wall, (i turn), and 4) making a 180° turn in a constricted area, ($U-$turn).

III. Reinforcement Learning Model

Within the project, two models were trained in simulation, one with off-policy Q-learning, and the other with on-policy SARSA. Only, the SARSA model was selected for sim-to-real transfer; however, support for demoing the Q-learning model is implemented. This is because the SARSA model learned a behavior that is more risk adverse and so was believed to be more robust against failure in the real world.

IV. Intrinsic Rewards for Safe Navigation

A significant problem in sim-to-real transfer is the difference between perfect movement in simulation vs. imperfect movement in the physical world. This could be as a result of wheel slipage or drift that wasn't accounted for in simulation. Because of this, it is possible for agents to learn behavior that works in simulation but is risky, dangerous, or less effective in the real world. To account for this, the reward structure is adapted before training that minorly punishes the agent for taking actions that are known to be risky or less robust in the real world. For example, turning to quickly. This punishment is not significant enough to discourage the agent from taking the actions if it needs to; however, will hopefully encourage it to select safer options when applicable. Through inspection of the real-world demo, the agent seemed to learn to travel straight most of the time which also improved the reliability/ repeatability of the system.

V. Domain Randomization

The intrinsic reward structure works well when there are known discrepancies between the simulation and the real world. However, there are likely still many unknown factors that cannot be accounted for in simulation. To account for this, random interference can be added to the simulation that will ideally enable the agent to learn behaviors that are more robust to unforeseen events. During training in simulation, minor random perturbations are added to the linear and angular velocities of each action before publishing to the robot. This means that after selecting an action, that action

will not be preformed exactly as expected. This discourages the agent from preforming behaviors that are too precise to work efficiently in the real world.

VI. Hardware Constraints

While demoing in simulation .sleep commands where added so that the navigation software wouldn't block other systems, and to prevent actions from being published too quickly. However, this was removed in the real world demo because the hardware on the robot may not be fast enough to resolve the overhead raised by the model between each time step. And so adding .sleep may cause delays between steps that can cause the robot to not get a correct action in time to avoid obstacles.

VII. Conclusion

Simulation is a powerful tool to support reinforcement learning; however, in for robotic models where the purpose is to represent complex behavior in the real world where can be many challenges when transferring from simulation to a physical system. This includes limitations in the simulation that prevent the agent from learning about novel experiences in the real world. To prevent this, an intrinsic reward and domain randomization were added during training. Generally, the physical robot performance closely matched the simulation and was able to complete all four behaviors without crashing into walls.