Using Reinforcement Learning with Real-Time Control of Stormwater for Protecting Downstream Ecosystems

Daniel Lassiter[1] and Jacob Nelson[1]

Abstract— **Stream habits across the United States are being impacted by various modifications made to the landscape, in many instances due to urbanization. In urbanized areas, the replacement of vegetation with impervious surface results in a dramatic shifts in hydrologic regime including increase in runoff volume and peak flow during rainfall events, potentially disrupting downstream biota. Typical stormwater storage and conveyance systems operate statically and are limited in their ability to match the pre-urbanized environment. In this study, we demonstrate the ability of real-time control (RTC) within a stormwater model to match undeveloped flow conditions using a policy derived from reinforcement learning (RL). Our primary contribution is in combining the Tensorforce and PySWMM python libraries to create an agent and environment for testing out reinforcement learning approaches. Preliminary results suggest that, with a little fine tuning, RTC from RL-derived policies could match undeveloped conditions as good as or better than traditional hydrograph matching approaches. Additional research is needed to find sets of agent and environment hyperparameters that yield the desired results.**

I. INTRODUCTION

More than 99 percent of streams and rivers have been impacted by human activity, often resulting in reduced diversity and ecological resilience [1]. Traditional stormwater systems which emphasize peak flow reduction can successfully mitigate some of the adverse affects of development-driven peak flow increases that can lead to erosion and degradation, but these static approaches ultimately contribute to reduced streamflow variability which can be problematic as well. Towards protecting downstream ecosystems, one fundamental stormwater management principle is to design stormwater infrastructure to mimic predevelopment flow regimes [2]. In practice, however, this can be difficult to achieve with a static system as flows are generally controlled by static orifices in which discharge only varies according to the depth of the corresponding storage facility.

This article investigates the promising solution of applying real-time control (RTC) with optimal control policies derived from reinforcement learning (RL) in order to match predevelopment flow regimes. In this approach, actuated valves are signaled to open and close based on an optimized RL-derived policy [3][4][5]. In addition to matching predevelopment flow regimes, this approach also has potential for increasing the capacity of the existing system and improving water quality [6].

[1]Daniel Lassiter, Department of Engineering Systems and Environment

[1]Jacob Nelson, Department of Engineering Systems and Environment

II. METHODS

A. The Model

The models that were developed to simulate developed and undeveloped conditions were modified from a model created by the developers of the python package Pystorms called scenario ’Theta’ for testing out real time control policies. Scenario Theta (Figure 1) includes two watersheds, two respective storage ponds with control orifices with actuated valves, and one main downstream pipe that receives the outflow from the ponds. This was modified in several ways. First, the imperviousness was increased to 40%. Next, the stage-storage curve for the ponds had to be modified to hold volumes that were more realistic for the runoff characteristics of the watershed. Afer that, the roughness and depression storage parameters had to be adjusted to more realistically reflect developed conditions. Finally, the model was duplicated to create an undeveloped scenario which had 0% impervious, no storage nodes, and

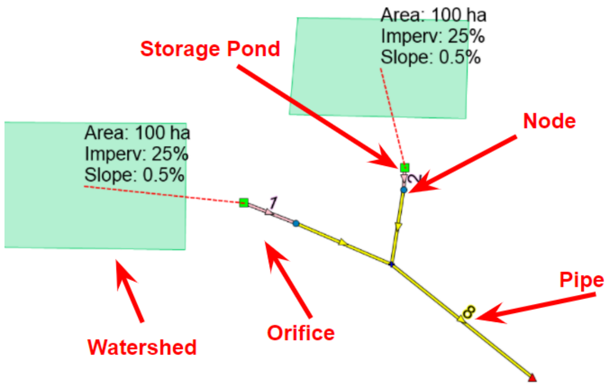

Fig. 1. Watershed catchment areas are equivalent in size, slope, and impervious area, though rainfall runoff generated from each site varies. (b) The flow passes from both watersheds through respective storage facilities, orifices, and pipes until converging downstream in Pipe 8.

roughness/depression storage parameters typical of forested conditions. The rainfall used to force both models was hourly rainfall data collected at the Norfolk Airport between 7/1/2013 and 10/1/2013.

B. *Python Packages*

The PySWMM package is a wrapper for accessing and running SWMM simulations which is written in FORTRAN [7]. The final setup of the environment and agent, as well as the training of the agent relied on Tensorforce, a python library constructed on Google's TensorFlow [8], which provides access to a number of configurable RL agents based on a variety of different RL algorithms. Pystorms, a library designed for testing out real time control techniques, was referenced for understanding how to implement PySWMM but the package was actually never loaded and used. Figure 2 shows how these packages wrap the model and each other.

C. *Specifying the agent*

The agent selected for use from the Tensorforce library is called, "Tensorforce". This agent is a general, configurable agent which acts according to a policy parameterized by a neural network [9]. The neural network we chose to have two layers each with 64 neurons. The Adam optimizer was also chosen which is used for stochastic gradient descent for training deep models and has been cited for being able to handle noisy gradients with little tuning required [10]. The Adam optimizer

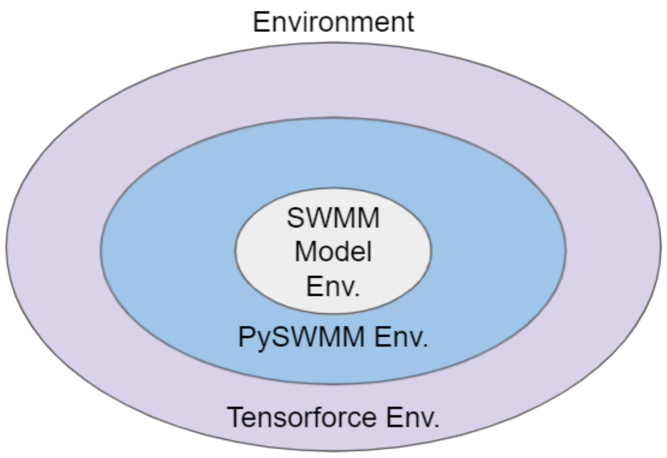

Fig. 2. PySWMM is a wrapper for the SWMM model which is written in FORTRAN, and the Tensorforce package relies on calls to classes that use PySWMM for running simulations and accessing outputs and modifying model parameters during the simulation.

is also able to handle large state spaces, though this was not a deciding factor for selecting the optimizer as the state space is relatively small, as described in the next section. We tried a tried a range of discount values ranging from 0.7 to 1 and we kept the learning rate constant at 1e-3.

D. *Specifying the Environment*

1) Overview: The environment with which the agent interacts is an object in the Tensorforce library called 'Environment.' The main parameters are the state space and actions space and then functions for taking actions, returning states and rewards, and resetting to run another episode. This required that we create our own environment class that relies on PySWMM functions. We designed this custom environment to be flexible enough to handle a range of state, action, and reward structure preferences as well as more complex models whose key attributes are specified in a YAML file loaded by each instance of the custom environment object.

2) State space: For this application, our state space included the flow in Pipe 8 (the downstream pipe receiving outflow from both ponds), the valve position in both of the upstream orifices, and the depth of the storage ponds. In addition to these five states, there are an additional 21 states containing values of pipe 8 flow and storage pond depth at lagged intervals of 1, 2, 4, 8, 16, 24, and 48 hours.

The purpose of this is to implicitly simulate an autoregressive forecast for the RL agent.

3) Action space: The action space for our agent was simply the valve position of the two orifices which could take a float value of 0 to 1 representing each orifice's fraction open. The agent was allowed to make an adjustment at intervals of 15-minutes which is also when it would receive rewards and the subsequent state.

4) Reward function: The purpose of the reward function is to help train the RL agent by penalizing decisions associated with undesirable outcomes with negative rewards and rewarding desirable outcomes with positive rewards. Recall, the objective is to obtain an optimal policy for mimicking undeveloped flows at an outfall of a stormwater system. We refer to these undeveloped flow as the "baseline" flow and they are one of three key variables for computing the reward. However, since in developed conditions there would likely be certain problematic flow thresholds that would be undesirable even if they occurred in baseline conditions, we've included a threshold value that causes steeper penalties when exceeded. This is the second key variable for computing the reward. Finally, if our agent for is within a pre-specified percent difference between baseline flows and controlled flows, it receives a positive reward. We chose 10% as this threshold and this represents the third key variable. The formula for computing the reward at each time step is shown below.

$$r = \begin{cases} 1/e^{p_d} & (p_d \geq t) \\ -(Q_c - Q_b)^2 & (Q_b \leq T, p_d > t) \\ -(Q_c - T)^2 & (Q_b > T, p_d > t) \end{cases}$$

$$p_d = |Q_c/Q_b - 1|$$

$$r = reward$$

$$Q_c = flow$$

$$Q_b = baseline\ flow$$

$$T = threshold\ flow$$

$$t = tolerance$$

E. Results

Our agent failed to learn the type of policy we were expecting. In each trial, we adjusted one or more hyperparameters and/or the reward function and the agent invariably converged on a policy that held the outfall at each pond either completely open or completely closed. The effect of these policies is nearly identical which is essentially to set the storage inflow equal to the outflow. The next two figures show the agents behavior after training for two episodes and then after training for seven episodes. The policy did not change from additional training. This was the trajectory of every training attempt in which various hyperparameters were modified such as the discount and horizon, the state space was modified (notably adding or removing the lagged variables), and the reward function was modified (method for calculating the positive reward and whether to include a positive reward at all).

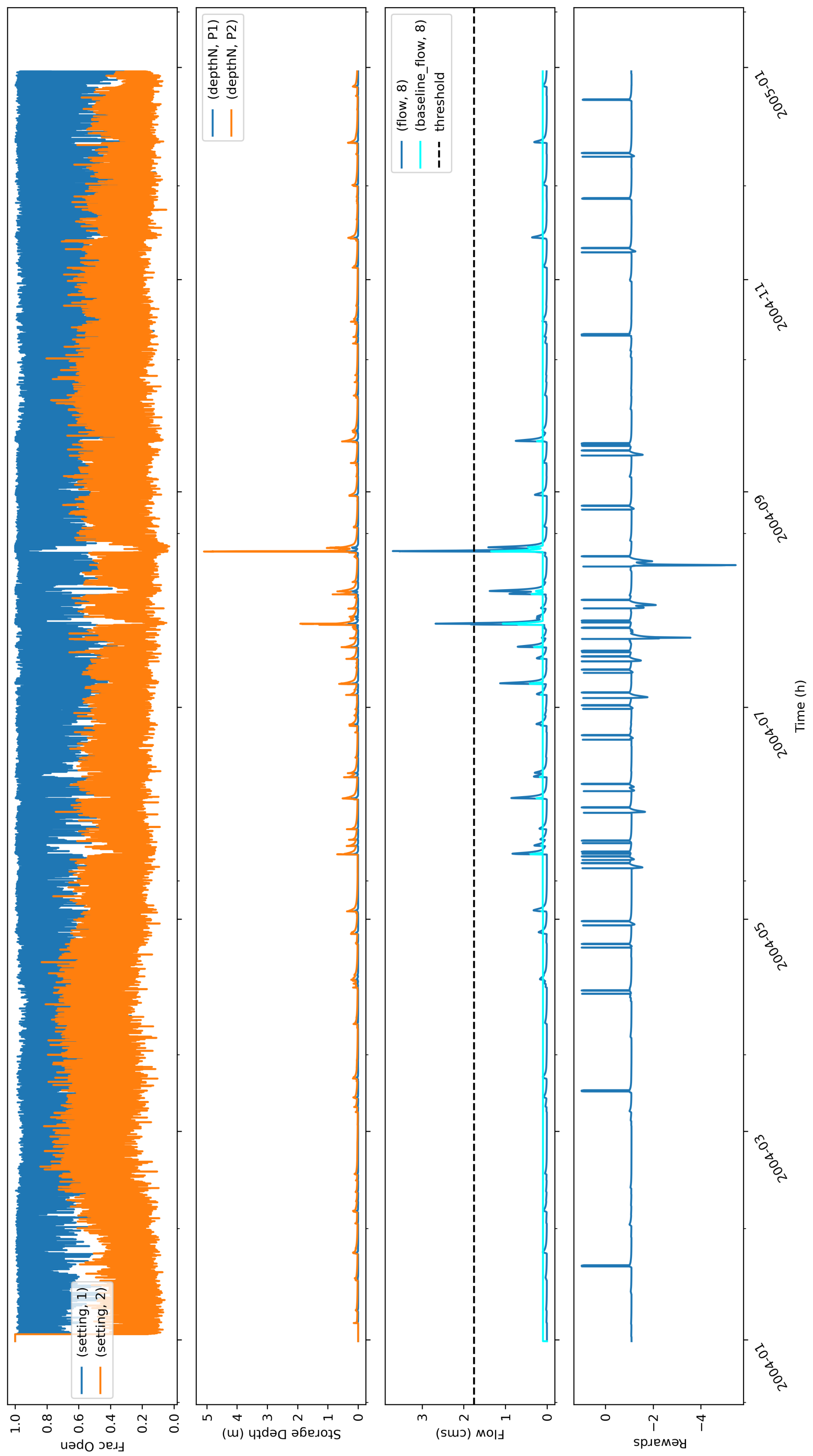

Fig. 3. Environment variables after 2 episodes of training. It's apparent that the agent has not yet converged on a stable policy.

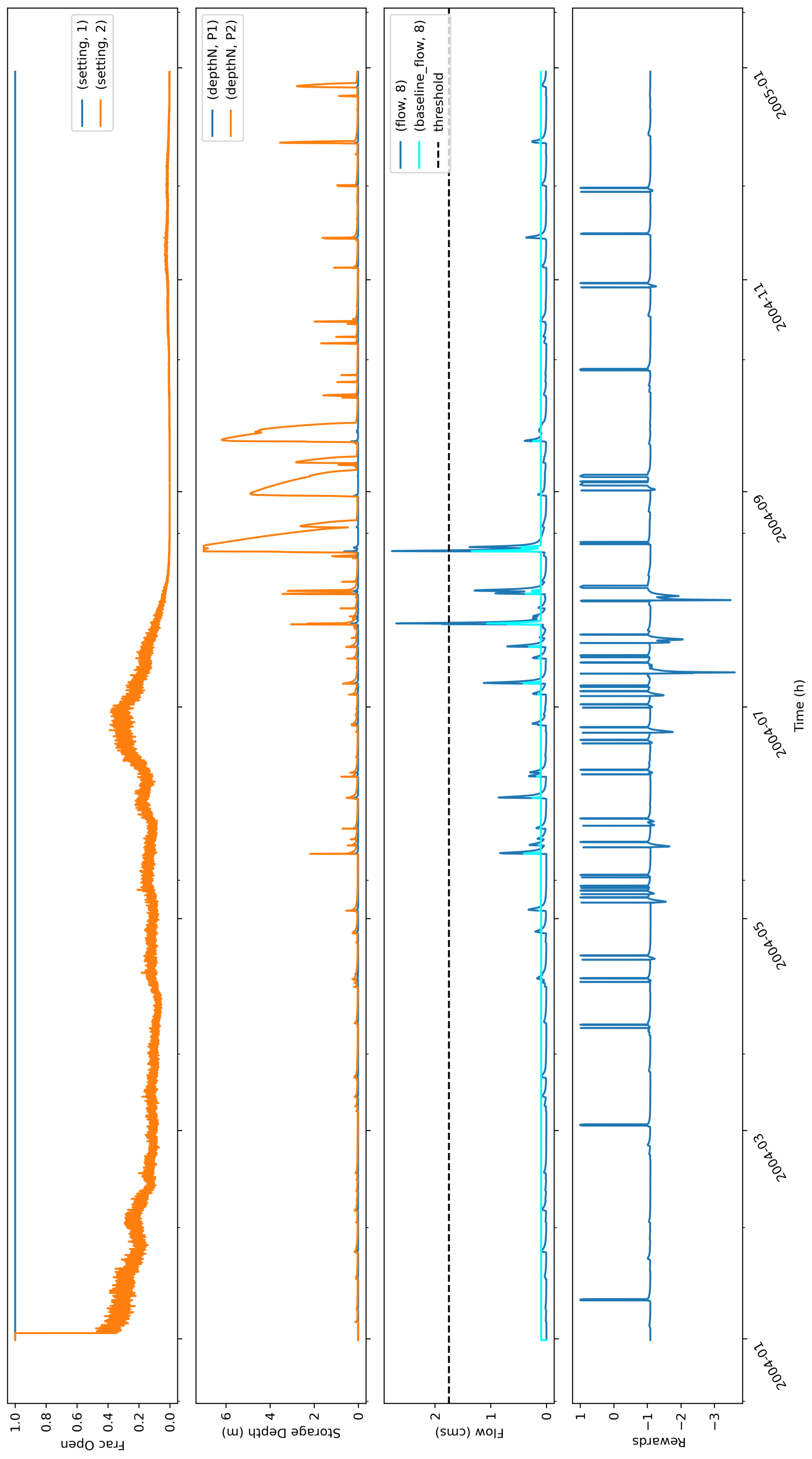

Fig. 4. Environment variables after 7 episodes of training. By the end of episode 7, the agent sets the orifice at pond 1 to be completely open and the orifice at pond 2 to be completely closed. This policy remains unchanged with further training. Comparing the scale of the reward function subgraph to that of the figure above, it is apparent that the magnitude of negative rewards is less so it's possible that this is in fact the optimal policy for the system.

III. Conclusion

While we were slightly disappointed not to achieve the type of policy we were looking for, we believe we are only a few small tweaks away. It is possible that the converged-upon policy is optimal for the environment and reward set up, so we may have been successful in that sense. We also successfully linked the PySWMM library with the Tensorforce library for testing out reinforcement learning for real time control and certainly identified environment and agent sets that fail to result in a more nuanced policy. Future work will include testing out alternatives to gradient descent and alternatives to neural networks for representing the policy and state spaces and in case the resulting policy was actually optimal, also modifying the environment such that an optimal policy better matches undeveloped flow regimes.

References

[1] Meter, K. Van, S. E. Thompson, and N. B. Basu. 2016. "Human Impacts on Stream Hydrology and Water Quality." In Stream Ecosystems in a Changing Environment, 441–90. Elsevier. https://doi.org/10.1016/B978-0-12-405890-3.00011-7.

[2] Walsh, Christopher J., Derek B. Booth, Matthew J. Burns, Tim D. Fletcher, Rebecca L. Hale, Lan N. Hoang, Grant Livingston, et al. 2016. "Principles for Urban Stormwater Management to Protect Stream Ecosystems." In Freshwater Science, 35:398–411. University of Chicago Press. https://doi.org/10.1086/685284.

[3] Bowes, Benjamin D., Arash Tavakoli, Cheng Wang, Arsalan Heydarian, Madhur Behl, Peter A. Beling, and Jonathan L. Goodall. 2020. "Flood Mitigation in Coastal Urban Catchments Using Real-Time Stormwater Infrastructure Control and Reinforcement Learning." Journal of Hydroinformatics, October. https://doi.org/10.2166/hydro.2020.080.

[4] Saliba, Sami M.; Bowes, Benjamin D.; Adams, Stephen; Beling, Peter A.; Goodall, Jonathan L. 2020. "Deep Reinforcement Learning with Uncertain Data for Real-Time Stormwater System Control and Flood Mitigation" Water 12, no. 11: 3222. https://doi.org/10.3390/w12113222

[5] Mullapudi, Abhiram, Matthew J. Lewis, Cyndee L. Gruden, and Branko Kerkez. 2020. "Deep Reinforcement Learning for the Real Time Control of Stormwater Systems." Advances in Water Resources 140 (June): 103600. https://doi.org/10.1016/j.advwatres.2020.103600.

[6] Zhang, Pingping, Yanpeng Cai, and Jianlong Wang. 2018. "A Simulation-Based Real-Time Control System for Reducing Urban Runoff Pollution through a Stormwater Storage Tank." Journal of Cleaner Production 183 (May): 641–52. https://doi.org/10.1016/j.jclepro.2018.02.130

[7] " Welcome to the Pystorms — Pystorms 0.0.2 Documentation." n.d. Accessed May 7, 2021. https://www.pystorms.org/docs/build/html/home.

[8] Schaarschmidt, Michael and Kuhnle, Alexander and Ellis, Ben and Fricke, Kai and Gessert, Felix and Yoneki, Eiko. 2018. "Reinforcement Learning in Computer Systems by Learning From Demonstrations." CoRR abs/1808.0. http://arxiv.org/abs/1808.07903.

[9] https://tensorforce.readthedocs.io/en/latest/agents/tensorforce.html

[10] Kingma, Diederik P, and Jimmy Lei Ba. 2015. "Adam: A Method for Stochastic Optimization." In 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.