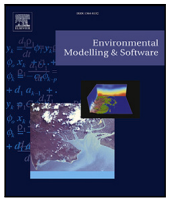
Environmental Modelling and Software 162 (2023) 105635

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft



Position Paper

pystorms: A simulation sandbox for the development and evaluation of stormwater control algorithms

Sara P. Rimer [a,1], Abhiram Mullapudi [b,1,2], Sara C. Troutman [b,1,2], Gregory Ewing [c], Benjamin D. Bowes [d], Aaron A. Akin [e,3], Jeffrey Sadler [d,4], Ruben Kertesz [f], Bryant McDonnell [f], Luis Montestruque [f], Jon Hathaway [e], Jonathan L. Goodall [d], John Norton [g], Branko Kerkez [b,*]

[a] Argonne National Laboratory, Decision and Infrastructure Sciences Division, 9700 Cass Avenue, Lemont, IL, 60439, USA
[b] University of Michigan, Civil and Environmental Engineering Department, 2350 Hayward Avenue, Ann Arbor, MI, 48105, USA
[c] University of Iowa, Department of Civil and Environmental Engineering, 300 South Riverside Drive, Iowa City, IA, 52245, USA
[d] University of Virginia, Department of Engineering Systems and Environment, 151 Engineer's Way, Charlottesville, VA, 22904, USA
[e] University of Tennessee, Department of Civil and Environmental Engineering, 851 Neyland Drive, Knoxville, TN, 37996, USA
[f] Xylem, Inc., 121 South Niles Avenue, 32, South Bend, IN, 46617, USA
[g] Great Lakes Water Authority, Detroit, MI, USA

ARTICLE INFO

Keywords:
Stormwater systems
Intelligent infrastructure
Adaptive control
Quantitative evaluation
Simulation sandbox
Open-source software

ABSTRACT

Advances in cyber–physical technologies have enabled real-time sensing and adaptive control of stormwater infrastructure. These smart stormwater systems allow for inexpensive, minimally-invasive stormwater control interventions in lieu of new construction. However promising the area of smart stormwater control, there still remain barriers – for experts and novices alike – to access shared tools and methods for investigating, developing, and contributing to it. In an effort to make smart stormwater control research more methodical and accessible, we present `pystorms`, an open-source Python-based simulation sandbox that facilitates the quantitative evaluation and comparison of control strategies. `pystorms` consists of a collection of real world-inspired smart stormwater control scenarios on which any number of control strategies can be applied and tested via an accompanying Python programming interface and coupled stormwater simulator. `pystorms` provides a framework for the rigorous and efficient evaluation of smart stormwater control methodologies across diverse watersheds with only a few lines of code.

1. Introduction

The advent of smart cities is poised to transform the management of our built environment (Harrison and Donnelly, 2011; Batty et al., 2012; Chourabi et al., 2012; Kitchin, 2014; Bibri and Krogstie, 2017; Eggimann et al., 2017). Specific to stormwater, a new generation of smart and connected stormwater systems aims to reduce flooding and improve water quality management by autonomously sensing watershed parameters and controlling hydraulic components across the watershed. These smart systems can provide an alternative to costly concrete-and-steel construction by squeezing even more performance out of existing stormwater infrastructure. Early ideas of controlling distributed stormwater systems in real-time date back to the 1970s (Trotta et al., 1977). The concept has, however, only recently gained widespread attention—in large part due to the affordability of internet-connected sensors, the increased capacity of data services, the broader emergence of other autonomous systems such as self-driving cars and robots, and the increasing prevalence of climate change stressors such as changing rainfall patterns and sea level rise. Relative to other fields of autonomy, however, smart water systems are still in the early stages of adoption. Thus, developing and implementing smart

* Corresponding author.
E-mail addresses: srimer@anl.gov (S.P. Rimer), abhiram.mullapudi@xylem.com (A. Mullapudi), sara.troutman@xylem.com (S.C. Troutman), gregory-ewing@uiowa.edu (G. Ewing), bdb3m@virginia.edu (B.D. Bowes), aakin@davisfloyd.com (A.A. Akin), jms3fb@virginia.edu (J. Sadler), ruben.kertesz@xylem.com (R. Kertesz), bryant.mcdonnell@xylem.com (B. McDonnell), luis.montestruque@xylem.com (L. Montestruque), hathaway@utk.edu (J. Hathaway), goodall@virginia.edu (J.L. Goodall), john@johnwnortonjr.com (J. Norton), bkerkez@umich.edu (B. Kerkez).
[1] These authors contributed equally to the paper.
[2] Present affiliation: Xylem, Inc., South Bend, IN, USA.
[3] Present affiliation: Davis & Floyd, Charleston, SC, USA.
[4] Present affiliation: United States Geological Survey, Middleton, WI, USA.

https://doi.org/10.1016/j.envsoft.2023.105635
Received 15 December 2022; Accepted 16 January 2023
Available online 25 January 2023
1364-8152/

water systems presents an exciting opportunity for researchers and practitioners to propose new visions, standards, and technologies.

The intelligence of smart stormwater systems broadly refers to the acquisition (i.e. "sensing") and processing of data into decisions and actions (i.e. "control strategies") that are then used to guide the operation of gates, valves, pumps, and other actuators within a watershed or drainage network. Ultimately, the logic embedded via these control rules determines how water is moved around the collection system to meet specific performance objectives, such as the reduction of flooding or improvement of water quality. Developing this control logic poses a major research frontier (Kerkez et al., 2016) and will require the engagement of groups and individuals from a wide range of experience and expertise.

Yet, entering this research field is presently precluded by a number of practical barriers. For new groups to make early contributions, they must have access to simulation testbeds, real-world inspired case studies, and appropriate control objectives. Due to privacy and safety concerns, access to stormwater network models and management details is difficult to come by without personal connections to those who manage local watersheds or municipal water systems. Thus, it can be difficult for new groups to obtain the necessary details of how real-world stormwater systems actually operate—leaving them to evaluate their control algorithms solely on idealized "toy problems". Though such idealized stormwater networks can aid in developing novel control algorithms, the applicability of these control algorithms in a physical system would require an evaluation in real-world-based stormwater networks. When access to such details is available, developing computational simulations that are true to real-world systems and objectives requires significant effort and expertise. Still, even when all of these other barriers are addressed and promising control algorithms have been proposed, they have usually all been evaluated on highly specific case studies and simulators, making it difficult to evaluate the extent of their success when applied to additional networks. In an effort to address these limitations, the contribution of this paper is `pystorms`, an open-source Python package comprised of:

(i) A collection of anonymized smart stormwater control scenarios that facilitate the quantitative evaluation and comparison of control strategies.
(ii) A programming interface and a stormwater simulator to provide a stand alone package for developing stormwater control strategies.

Our aspiration is for `pystorms` to emerge as a community-driven resource that fosters accessibility and collaboration amongst researchers and practitioners, both novices and experts alike. To facilitate broader collaboration and accelerated adoption, this paper is accompanied by an expanded supplementary information, online user guide,[5] and code to allow others to adopt the toolbox to their own control problems.

2. Background

2.1. Control of stormwater systems

A stormwater control problem can be defined as finding a strategy to manipulate the flow of stormwater to achieve a desired water quantity or quality objective. Traditionally, stormwater control has relied on *passive* solutions, in which control is achieved via large-scale and expensive construction. Instead, the emergence of microcontrollers, wireless communication technologies, and low-cost sensors has allowed for *active* solutions, in which existing infrastructure can be retrofitted with low-cost wireless valves, pumps, and other actuators, installed at strategic locations throughout a stormwater network, and utilized adaptively, in *real-time*. Consequently, stormwater infrastructure can now be instantly reconfigured to respond to its dynamic environment.

While real-time stormwater control engineering solutions were documented at least a decade earlier (Trotta et al., 1977), research-oriented discussion of these implementations did not occur until 1989 (Schilling, 1989). Furthermore, while limited implementation of smart stormwater control began at the end of the 20th-century, the 21st-century has seen far more extensive and systematic successes, as described in the foundational reviews of Schütze et al. (2004) and Vanrolleghem et al. (2005). Some notable smart stormwater control implementations include Ocampo-Martinez (2010), Gaborit et al. (2013), Vezzaro and Grum (2014), García et al. (2015), Gaborit et al. (2016), Mullapudi et al. (2017), Montestruque (2018), Sadler et al. (2019), Persaud et al. (2019), Bowes et al. (2021). These references contain instances of smart stormwater control from single control assets to watershed-scale implementations. For more comprehensive reviews of stormwater control implementations, we direct the reader to some recently published survey articles on the topic (Yuan et al., 2019; Lund et al., 2018; Shishegar et al., 2018; van Daal et al., 2017; García et al., 2015).

2.1.1. Simulating stormwater systems

Due to safety concerns and the variability of storm events, it is often infeasible to test various control strategies on actual stormwater networks. A more practical approach is to first estimate the outcomes of different control decisions in a computer simulated environment , before deciding to port them to real systems. These simulations can be carried out using a computational stormwater model. The components of such a model usually include a (i) runoff module and (ii) a routing module, and are driven by (iii) precipitation events (e.g. rain, snow). The runoff module converts precipitation into overland runoff; the overland runoff then undergoes hydrological processes (e.g. infiltration, evaporation) and is hydraulically transported to the stormwater collection system.

Over the years, several different software applications have been developed for modeling and simulating stormwater networks. The different software applications all function in a similar manner: they compute the dynamics of stormwater as it moves through its local watershed. The US-EPA's Stormwater Management Model (SWMM) (Rossman, 2015), MIKE URBAN+ from the MIKE Powered by DHI software suite,[6] and the Model for Urban Stormwater Improvement Conceptualisation (MUSIC) by eWater[7] are examples of popular stormwater software applications. The theory and computational details of these models are summarized in Rossman and Huber (2015), Rossman (2017), Rossman and Huber (2016). While these models provide powerful simulation adroitness for modeling hydraulic and hydrologic phenomena, they confine real-time control to limited rule-based approaches, thus limiting their out-of-box use in the study of smart stormwater systems.

2.1.2. Implementing adaptive control

Here, we aim to maintain the idea of control in its broadest but most straightforward meaning: after receiving some sort of *cue* (from a sensor or state estimate), an *action* is taken (for instance opening or closing a valve) with the aim of achieving a desired outcome (such as reducing flooding or improving water quality). When we implement *control*, we are deciding on the course of action for optimizing our system to meet some specified objective. A stormwater control strategy seeks to formalize this process of deciding a set of actions. We define the computational process of implementing this strategy as the *stormwater control algorithm*.

Suppose a stormwater system with only one valve was installed, and that valve can either be completely opened or closed every hour.

[5] pystorms.org/docs

[6] mikepoweredbydhi.com

[7] ewater.org.au

Deciding on a pattern for the complete opening and closing of the valve, over the period of a few hours to even a few days, presents a multitude of permutations and is a non-trivial undertaking. By expanding the task to allow for the valve to opened at any number of percent increments between 0%–100%, the combination of actions that can be implemented becomes even more expansive, making simple rule-based or "if-else" algorithms either ineffective or bafflingly large. This complexity is particularly true for more sizable drainage systems, where tens to hundreds of valves need to be operated.

Hence, finding the "best" control actions is difficult and – by nature – subjective. Furthermore, most algorithms are often only evaluated on a single case study, and only in comparison to an uncontrolled solution. While valuable, this has the effect of making most algorithms appear very effective, since the only baseline that exists otherwise is the unoptimized, "as-built" static system.

2.2. The need for a simulation sandbox

For new researchers and practitioners in the field of smart stormwater control, the barrier to entry is significant. Newcomers must spend considerable time searching for real-world case studies, synthesizing relevant control objectives, developing algorithms, and building the capacity to carry out simulation in order to test new ideas. To grow this research field and support its broader scientific context, there is a need to better enable the cross-comparison of smart stormwater control strategies, their algorithms, and the case study instances for when they are used. While there have been some prior efforts to benchmark specific stormwater networks in order to evaluate adaptive control strategies (Schütze et al., 2017; Borsányi et al., 2008), there is still a shortage of stormwater control case studies with diverse control objectives.

Other research domains provide compelling examples of community-driven simulation tool chains that have been developed for similar cross-comparison needs. For example, the ARPA-E GRID DATA program has enabled an active research community in the energy sector by providing open source case studies and benchmarking tools of power system networks (Advanced Research Projects Agency–Energy, U.S. Department of Energy, 2016). In a similar vein, the water distribution community has created their own active cross-comparison tools (Walski et al., 1987; Ostfeld et al., 2008; Marchi et al., 2014) and websites.[8] The hydrology community has also developed a watershed-scale hydrometeorological dataset and model performance benchmarks for evaluating hydrologic models (Newman et al., 2015). `pystorms` is our effort to develop a similar set of research tools for the stormwater control community. Additionally, we have also been inspired by the streamlined control algorithm testing of the OpenAI Gym toolkit (Brockman et al., 2016). As such, we contend that there is a need for a similar, more "out-of-the-box" software toolset with an unambiguous programming interface that allows stormwater control researchers to get started more quickly. To that end, we have formulated `pystorms` as a *simulation sandbox*, in which we systematize a collection of stormwater control testbed examples, and foster the experimentation and testing of new control strategies.

3. Pystorms

To achieve the objectives of curating an open repository of smart stormwater control testbed examples, and reducing the learning curve for newcomers testing new control strategies, `pystorms` is underpinned by two distinct features. First, it provides a collection of diverse stormwater control scenarios, which are drawn from real-world urban watersheds to encompass diverse features pertaining to stormwater systems (Section 3.1). Second, these scenarios are coupled with a streamlined Python programming interface (Section 3.2) that explicates the computational backend of a corresponding stormwater control simulator (Section 3.3). Together, these features provide researchers with a standalone software package that focuses its usage on the development and testing of stormwater control algorithms.

3.1. Scenarios

`pystorms` abstracts smart stormwater systems into *scenarios*. Each scenario captures a combination of elements that comprise a stormwater control problem. A fully defined scenario includes a network topology (the system or watershed being studied), inputs, a selection of controllable and observable assets, as well as a clearly defined control objective (we refer the reader to Fig. 1 and Table 1 for further details). While users can – and are encouraged to – create their own scenarios, at the time of writing, `pystorms` provides an initial collection of seven scenarios, all drawn from real-world smart stormwater systems across North America and Europe. The collection of scenarios spans a variety of stormwater systems that address a diverse set of urban watershed needs with various control objectives. The subcatchment areas range from 0.12–$67\,\mathrm{km}^2$ in size, and include both combined and separated stormwater arrangements. A summary of the collection of scenarios are presented in Table 2, with their more detailed descriptions provided throughout this paper's supplementary documents.

While our aim is for this collection of scenarios to represent a myriad of smart stormwater control applications, we recognize that it is certainly not exhaustive. Ultimately, we aspire to grow the `pystorms` repository of stormwater scenarios through community-driven contributions of new scenarios. Accordingly, we provide extensive documentation[9] for users to contribute their own scenarios, or modify the existing ones.

3.2. Workflow

`pystorms` provides a suite of pre-defined smart stormwater systems, and is designed to be both intuitive and accessible for users at any level of expertise or experience of control systems and stormwater management. Via three intuitive function calls, the user is able to iterate through the simulation of a smart stormwater scenario, and interact with the scenario at any of the simulation timesteps by querying its states or changing the settings of its control assets. `pystorms` provides the computational environment of a smart stormwater system; the user then provides the control algorithm dictating how it should operate. For developing control strategies, `pystorms` allows users the flexibility to utilize any additional computational tools at their disposal. This can be done either by leveraging any additional computational software stacks of Python, or interfacing `pystorms` with other computing platforms or languages (e.g., MATLAB, Julia).

The user first initializes a `pystorms` scenario by creating an instance of it using the statement: `pystorms.scenarios.<scenario name>()`. As seen in the code example (Fig. 2), `theta` can be initialized with `pystorms.scenarios.theta()`. The initialization configures the stormwater simulator with the computational representations necessary to simulate the respective scenario, and returns it as a Python object. This Python object (`env` in Fig. 2) can be used to progress and/or pause the stormwater simulator, read and/or write parameters to the network, and utilize any additional `pystorms` functionality.

The `pystorms` programming interface is inspired by the principles of control theory, where the control of a system is abstracted as a control loop in which a controller monitors the underlying state(s) of the system and makes calculated adjustments to the system for it to

[8] uknowledge.uky.edu/wdsrd, wateranalytics.org/EPANET

[9] pystorms.org/docs

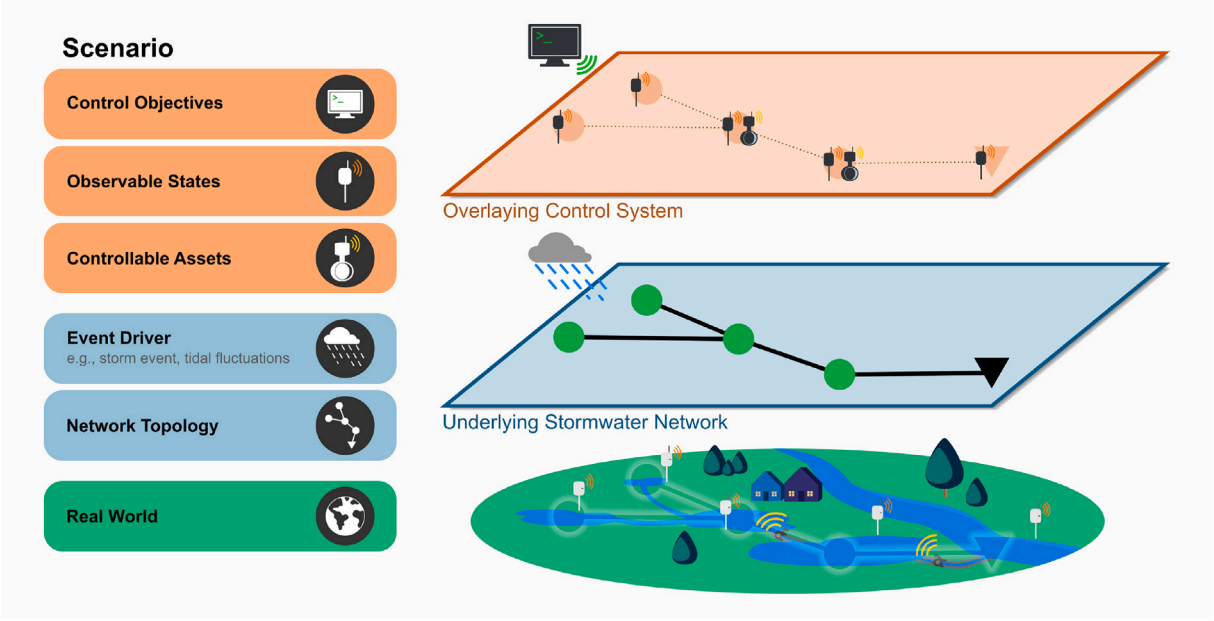


Fig. 1. `pystorms` abstracts the control of stormwater systems as scenarios, each characterized by an overlaying control system and an underlying stormwater network. The overlaying control system (in orange) encapsulates what can be considered the stormwater control system's virtual elements – that is, the components of the stormwater control system that are changeable and/or readable by an implemented control algorithm – and include its Control Objectives, Observable States, and Controllable Assets. The underlying stormwater network (in blue) represents the computational implementation of the hydraulic and hydrological elements of the stormwater control system, namely its Network Topology and Event Driver. Table 1 presents a detailed description of these elements.

Table 1
A `pystorms` scenario is comprised of five distinct components: its Network Topology, an Event Driver, a set of Controllable Assets, a set of Observable States, and the Control Objectives.

Network topology	A *network* is the physical system of conduits (e.g. pipes, culverts), storage elements (e.g. retention and detention basins), and any other subcatchment infrastructure (e.g. green infrastructure, wetlands) that collect, convey, and/or treat stormwater runoff.
Event driver	The *event driver* consists of any inputs or "disturbances" to the network that govern the generation and flow of runoff. Most often, an event driver is the precipitation generating runoff in the watershed, but can also include wastewater flows, tidal fluctuations, or other phenomena.
Controllable assets	The *controllable assets* are the subset of the network topology that are equipped with valves, pumps, or any other flow control infrastructure that can be actuated to manipulate stormwater flow.
Observable states	The *observable states* are the collection of states in the network that can be accessed by the users during a simulation. In the real world, these states are measured by a set of sensors installed at the corresponding network locations.
Control objectives	The overall goal or set of goals (e.g. preventing flooding, reducing erosion, minimizing overflows, improving water quality) of manipulating the behavior of a stormwater network using controllable assets during a simulation. A control strategy's ability to achieve a particular objective is quantified via a corresponding performance metric.

achieve a desired behavior. The basic control loop is implemented using the following steps:

(1) **Query the set of observable states** in the stormwater network at the current time-step;
(2) **Compute control actions** to manipulate the system to achieve a desired behavior; and
(3) **Implement the control actions** by adjusting the settings of the controllable assets.

The state of the underlying stormwater network in the scenario can be queried using the corresponding method (Fig. 2, line 15). A stepping method (Fig. 2, line 19) implements the control actions in the stormwater network, progresses the simulation forward a time-step, and returns the current status of the simulation, terminating using a logical operator. The stepping method also implements `actions` in the stormwater network and progresses the simulation being handled by the `environment` object, which in this case is the Scenario `theta` (Fig. 2, line 10). `done` is assigned `True` when the simulation has terminated, and `False` otherwise.

During the each time-step of the simulation, the metrics that underlie the scenario's control objective are evaluated. This computed value is then stored for each time-step, and can be accessed at any time during the simulation using the performance method (Fig. 2, line 22). Additional parameters are logged throughout the simulation. While an initial set of these logged parameters is predefined, the user is able to customize this set for any additional parameters of interest. The user defines their controller using a custom method (Step (2)), which maps the observed states to control actions (Fig. 2, line 4).

3.3. Architecture

The `pystorms` architecture follows an object oriented programming paradigm and relies on classes as its core building blocks. While the `pystorms` programming interface is designed with the intent to

Table 2
`pystorms` includes a curated collection of real world-inspired stormwater scenarios. Users implement their own control algorithms.

Scenario	Network topology	Control objectives
`alpha`	0.12 km^2 residential combined sewer network	Minimize total combined sewer overflow volume (5 weirs at interceptor connections)
`beta`	1.3 km^2 separated stormwater network with a tidally-influenced receiving river	Minimize flooding (1 storage basin outlet, 1 pump, 1 inline storage dam)
`gamma`	4 km^2 highly urban separated stormwater network	Maintain channel flows below threshold and avoid flooding (11 detention pond outlets)
`delta`	2.5 km^2 combined sewer network in which the stormwater ponds also serve as waterfront	Maintain water levels within upper and lower thresholds for water quality and aesthetic objectives (4 storage basin outlets; 1 infiltration basin inlet)
`epsilon`	67 km^2 highly urban combined sewer network	Maintain sewer network outlet total suspended solids (TSS) load below threshold and avoid flooding (11 in-line storage dams)
`zeta`	1.8 km^2 combined and separated sewer network (based on the Astlingen benchmarking network (Schütze et al., 2017; Sun et al., 2020))	Maximize flow to downstream wastewater treatment plant and minimize total combined sewer overflow volume (4 storage basin outlets)
`theta`	2 km^2 idealized separated stormwater network	Maintain the flows at the outlet below a threshold and avoid flooding (2 storage basin outlets)

be intuitive for all potential users, it particularly caters to those who may only have a rudimentary understanding of stormwater dynamics and/or basic familiarity with programming in Python. However, it can also be customized to meet the requirements of researchers who want to incorporate advanced functionality, such as custom water quality or rainfall-runoff modules. For details on how to utilize `pystorms` modularity and customization, we direct the reader to its online documentation.[10]

The `pystorms` architecture is organized to accomplish two tasks: (1) configure the `pystorms` scenario, and (2) execute the `pystorms` scenario. These two tasks are carried out using three core interacting modules: `environment`, `scenario`, and `config`. These three modules interface with each other to build and execute a given scenario. Fig. 3 provides a schematic of this architecture. The first two modules handle the stormwater simulation, while the latter handles the computational representation of the stormwater networks and the metadata pertaining to the control problem (i.e. states, actions, and objectives).

3.3.1. Configuration

The `config` module is used to manage the configuration in the `pystorms` architecture. `config` contains a configuration file for each scenario, which specifies the stormwater network, and then delineates its observable states, controllable assets, and the set of parameters that are used to compute its control objective's corresponding performance metric. The configuration files are written using YAML, a mark-up language commonly used for developing configuration files in software applications. With YAML, the parameters of interest defined in the configuration file are formatted as vertical lists rather than data structures. As a result, the configuration file becomes more human-readable, and creates a simple but scalable workflow for developing scenarios. Example YML files are provided in the supplementary information of this paper and the online guide.

[10] github.com/kLabUM/pystorms

3.3.2. Simulation

Scenarios in `pystorms` are implemented as Python classes. To ensure consistent functionality across scenarios, each scenario is instantiated as its own independent class with an inherited structure from a base `scenario` module. The scenario classes interface their corresponding configuration files with the stormwater simulator and implement any of the functions specific to that scenario (e.g. functions used for computing performance metrics of corresponding control objectives).

The `environment` module is the interface between the stormwater simulator (e.g. EPA-SWMM) and the scenarios. This module is specifically included to ensure `pystorms` is able to remain agnostic to whatever stormwater simulator is used. For instance, if a user wants to utilize a customized hydrologic solver for simulating stormwater, they can do so by modifying the `environment` module to call their solver when the scenarios query it, thus ensuring compatibility to a wide array of simulators with minimal overhead.

`pystorms` uses SWMM as its default stormwater simulator. SWMM, developed by the U.S. EPA, is an open-source stormwater simulation model that is extensively used for the design and analysis of stormwater systems across the world. Despite SWMM's prevalence in stormwater modeling, it has limited capability to simulate green infrastructure, pollutant transformations, and overland flooding. SWMM is written in C, a low-level programming language that results in significant computational efficiency. However, the tradeoff for using C becomes SWMM's subsequent difficulty at being interfaced with the latest scientific libraries, which are primarily developed in high-level programming languages, such as Python. As a result, there have been several efforts over the years to build wrappers for SWMM such that its functionality can be exploited via these high-level languages. High-level programming languages wrappers for SWMM have enabled the creation of tools that enhance SWMM's ability to model complex phenomenon in

PySWMM is a Python package that not only provides a wrapper to communicate with SWMM, but also yields a high-level user interface for

```python
import pystorms


def controller(state):
  # your control algorithm goes here
  return actions


# initialize scenario
env = pystorms.scenarios.theta()
done = False

while not done:
  # query current state
  state = env.state()
  # compute the control action
  actions = controller(state)
  # implement the action
  done = env.step(actions)

# check your controller's performance
env.performance()
```

Fig. 2. This code snippet is an example implementation of `pystorms` for Scenario `theta`, which is described in greater detail throughout Section 4. In this example, the controller is implemented as a Python function block.

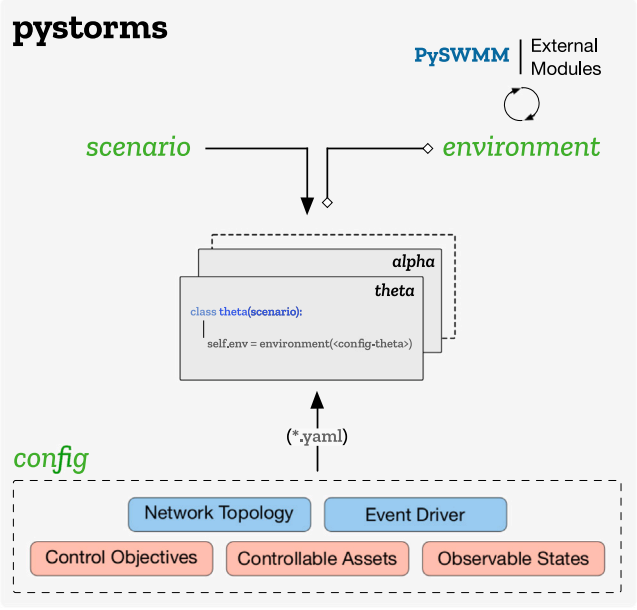


Fig. 3. `pystorms` is built with three interacting core modules: (i) `config` represents the metadata and computational representations of the stormwater network and event driver; (ii) `environment` acts as an interface for scenarios to interact with the stormwater simulators; and (iii) `scenario` provides a consistent structure for the scenarios in the package. A scenario object in `pystorms` inherits (represented by arrows) from the base `scenario` class, and interfaces (represented by the line) with the stormwater simulator though the `environment`.

querying the various stormwater parameters (McDonnell et al., 2020). `pystorms` – by means of the `environment` module – interfaces with SWMM using PySWMM, and as a result, all functionality included in PySWMM can also be accessed using `pystorms`. Readers are directed to the documentation[11] for additional details and examples to customize `pystorms` to meet their requirements.

3.4. Software availability

Developed in Python, `pystorms` is supported on all major operating systems (OSX, Windows, and Linux) and can be installed using `pip`.[12] `pystorms` is distributed under the GNU General Public GPLv3 license,[13] which ensures that this package and its derivatives remain open-source and can be used free of cost. Additionally, source code for the package is available on Github.[14] alongside comprehensive documentation and tutorials to utilize and contribute to its broader utilization and development[15]

4. Demo: Implementing and evaluating control strategies

Here, we demonstrate how `pystorms` facilitates developing smart stormwater control strategies by evaluating the performance of two control algorithms applied to Scenario `theta`.

4.1. Scenario *theta*

To demonstrate a simple example of a pystorms scenario, we can focus on Scenario `theta`, an idealized stormwater network synthesized for unit testing and rapid algorithm exploration. Scenario `theta`'s *network topology* includes two $1000\,m^3$ storage basins connected in parallel and draining into a shared downstream water body. The *event driver* is a synthetic rain event lasting 9 hr with a peak intensity of $3.2\,in\,hr^{-1}$. The *observable states* are the water levels at the two basins, reported at 15 min time-steps. DRAFT: `theta` is simulated at 30 sec interval in to better represent the dynamics of the stormwater network. The *controllable assets* are valves at the outlet of both storage basins, adjustable at each time-step between $0-100\,\%$ open. The *control objective* is to maintain the outflow into the downstream water body below a specified threshold of $0.5\,m^3\,s^{-1}$, while simultaneously preventing flooding at the basins. The ability of a control strategy to meet `theta`'s control objective is quantified using a pre-defined performance metric that computes a penalty for violating the control objective at each time-step, and sums these penalties across the whole simulation. We provide the specific details on this performance metric (Eq. (1a)) in Section 4 where we evaluate the performance of two different example control strategies applied to `theta`.

Scenario `theta` has been developed for rapid prototyping and unit testing of new control strategies. Because Scenario `theta` is an idealized case, its corresponding performance objective is defined such that a "perfect" score of 0 is achievable. We demonstrate how to utilize Scenario `theta` in this manner by presenting two different control strategies applied.

4.1.1. Implementing control strategies

While there exist many control strategies that can be adopted to achieve `theta`'s control objective, we implement and compare two basic strategies here: a simple rule-based control strategy, and the Equal-filling Degree control strategy. Both control strategies adjust the valve openings of `theta`'s two basin outlets to either retain or release

[11] github.com/kLabUM/pystorms
[12] pypi.org/project/pystorms
[13] gnu.org/licenses/gpl-3.0.html
[14] github.com/kLabUM/pystorms
[15] pystorms.org

Algorithm 1: Equal-Filling Control Algorithm: Let i be a tank in the network of $\mathcal{N}$ tanks. In scenario theta, $\mathcal{N} = 2$ and Max depth in each tank is 2.0m

Let λ be the target flow to be achieved
for *all* $\mathcal{N}$ *tanks* **do**
  Compute the *filling degree*; $f_i = \text{depth}_i / \text{Max depth}_i$
Estimate the *average filling degree*; $\bar{f} = \sum_i^N f_i / N$
for *all* $\mathcal{N}$ *tanks* **do**
  Let $\psi_i = f_i - \bar{f}$
  if $\psi_i < 0.0$ **then**
    $\psi_i = 0.0$
  else if $\psi_i = 0.0$ **then**
    $\psi_i = \bar{f}$
for *all* $\mathcal{N}$ *tanks* **do**
  Assign valve positions; $v_i \propto \lambda \times \{\psi_i / \sum_i^N \psi_i\}$

```python
def controller(
    depths,
    tol = 0.50,
    LAMBDA = 0.50,
    MAX_DEPTH = 2.0):

    # Compute the filling degree
    f = depths / MAX_DEPTH

    # Estimate the average filling degree
    f_mean = np.mean(f)

    # Compute psi
    N = len(depths)
    psi = np.zeros(N)
    for i in range(0, N):
        psi[i] = f[i] - f_mean
        if psi[i] < 0.0 - tol:
            psi[i] = 0.0
        elif psi[i] >= 0.0 - tol and psi[i] <= 0.0 + tol:
            psi[i] = f_mean

    # Assign valve positions
    actions = np.zeros(N)
    for i in range(0, N):
        if depths[i] > 0.0:
            k = 1.0 / np.sqrt(2 * 9.81 * depths[i])
            action = k * LAMBDA * psi[i] / np.sum(psi)
            actions[i] = min(1.0, action)
    return actions
```

Fig. 4. The Equal-filling Controller maintains the flows at the outlet below a desired threshold by coordinating its actions such that it equally utilizes the storage in the controllable assets of the network. Algorithm 1 and the corresponding code snippet illustrate the algorithm and its implementation as a function block in Python. An interactive example of the algorithm implementation and its evaluation on Scenario `theta` can be accessed at pystorms.org/#ipynb/GoogleColab/theta.

storage depending on the observed states; however, the rule-based control strategy illustrates a simple example that may be explored by a first time user, while the Equal-filling Degree control strategy is presented as an example of an established methodology widely used by stormwater control practitioners.

Rule-based controller

The rule-based control strategy adjusts our basin outlets based on their respective water levels. Specifically, each basin's outlet setting is equal to its relative water level (i.e., the current water level of the basin divided by its maximum depth). Therefore, our control algorithm will set a full basin's outlet to 100% open, and a basin that is half full will have its outlet set to 50% open, etc. While this strategy provides a means to mitigate local flooding at each basin, it notably does not consider the other control objective for the network's outflow into the downstream water body to stay below a given threshold. The python code for this simple rule-based controller is shown in the supplementary information of this paper, while the code for a more complex control follows in next section.

Equal-filling degree controller

The equal-filling degree is a control strategy often applied to stormwater networks with distributed stormwater storage assets, and has been used in some cases as a starting point when comparing more than one control strategies (Borsányi et al., 2008; Campisano et al., 2000; Dirckx et al., 2011; Kroll et al., 2016; Vezzaro and Grum, 2014). For this strategy, we begin by defining a storage asset's "filling degree" – which is typically the ratio a storage asset is full based on its volume or depth – and compute it for each asset in the collection system. The algorithm seeks to "balance" these filling degrees across the system based on its average. The exact manner in which this balancing is carried out is not necessarily consistent in literature. Our method for this balancing is delineated in the algorithm in Fig. 4. If all assets have a filling degree equal to the average (i.e., all assets are equally filled), then each should release an equal fraction of the target outflow. Otherwise, the released flows across the assets should be differentiated such that, when an asset has a filling degree less than the average, it does not release any flow; but if an asset is greater than the average, it releases flows based on its deviation from the average.

The implementation of the equal-filling degree algorithm using `pystorms` can be seen in Fig. 4. We carry out the simulation for each of the two algorithms, as well as for the *uncontrolled case*, in which control actions are never implemented and the basin outlets are always open. The resulting hydraulic behavior at the two basins and the network's outflow for each of these simulation runs can be seen in Fig. 5.

4.1.2. Evaluating control strategies

The aim of Scenario `theta` is to find a control strategy that can meet `theta`'s *control objective* to maintain the outflow into the downstream water body below a specified threshold of $0.5\,\text{m}^3\,\text{s}^{-1}$ and also minimize flooding at the basins. As discussed in Section 3.1, we predefine a performance metric to quantify the control algorithm's ability to meet the corresponding control objective. For Scenario `theta`, this performance metric, P, is defined as:

$$P = \sum_{t=0}^{T} \left(\mathcal{H}_t + \sum_{i=1}^{2} \mathcal{G}_{i,t} \right) \tag{1a}$$

$$\mathcal{H}_t = \begin{cases} Q_t - 0.5\,, & \text{if } Q_t > 0.5 \\ 0.0\,, & \text{otherwise} \end{cases} \tag{1b}$$

$$\mathcal{G}_{i,t} = \begin{cases} 10^3\,, & \text{if any flooding at basin } i \\ 0.0\,, & \text{otherwise} \end{cases} \tag{1c}$$

where $\mathcal{H}_t$ is a flow exceedance penalty of the stormwater network's outflow, Q_t, over the $0.5\,\text{m}^3\,\text{s}^{-1}$ threshold; and $\mathcal{G}_{i,t}$ is an arbitrary flooding penalty of 10^3 added whenever there exists flooding at either of our two basins, both calculated and summed across every time-step t in the simulation. The performance metric, which is calculated across the simulations for the uncontrolled case and both of the implemented control algorithms, can be seen in Table 3. Additionally, corresponding hydraulic behavior for all three cases at their network outlet and both basins can be seen in Fig. 5. The equal-filling degree strategy is able to achieve the control objective of the outflow threshold, as well as avoidance of flooding. Alternatively, the rule-based control strategy only is able to avoid flooding at the basins. The stormwater network behavior for both strategies follow their corresponding implemented algorithm. For example, as the rule-based control strategy does not directly consider the outflow threshold when determining the implemented control actions, it follows that the outflow in the network's outlet exceeds this threshold (see the outlet plot in Fig. 5). To support variety of control algorithms, `pystorms` uses the general concept of a performance metric in lieu of a formal control objective and cost function. Depending on the choice of control algorithm, users can create their control strategies and cost fictions to maximize the overall performance. For example, while equal-filling may implicitly result good implicit performance, the explicit control objective of equal-filling is to maintain water level throughout assets.

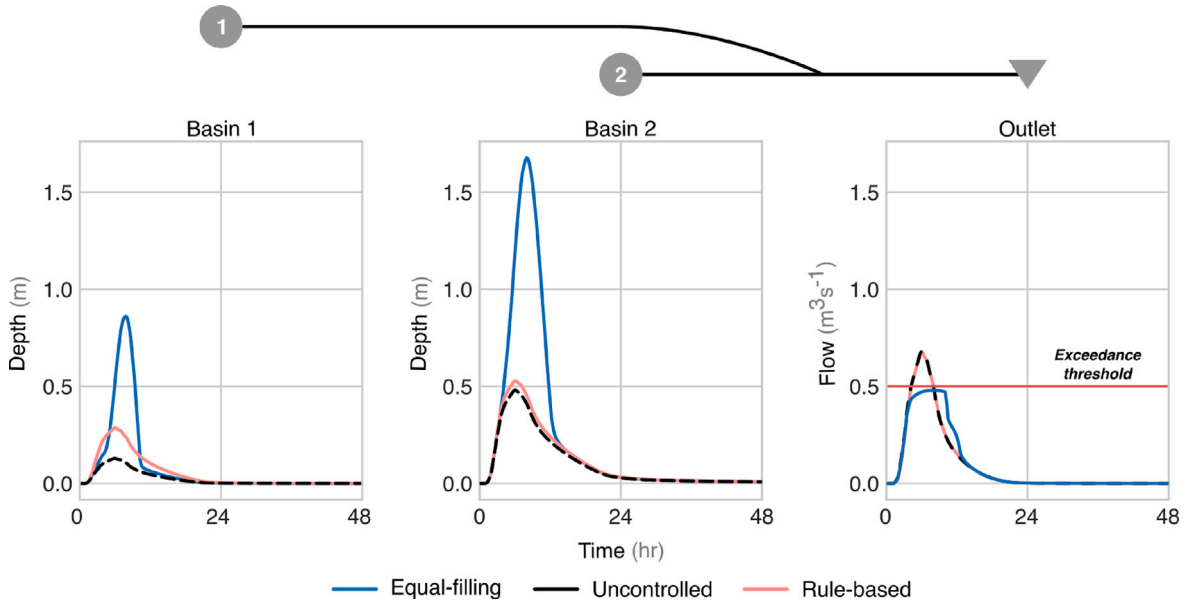


Fig. 5. In Scenario `theta`, the equal-filling degree control strategy is successfully able to maintain the flows at the outlet of the watershed below the desired threshold of $0.5\,m^3\,s^{-1}$ by uniformly using the storage in the networks. Static rule-based control and uncontrolled responses of the networks are also presented for comparison. The maximum depth in each of the two basins is 2 m. The simplified network topology of Scenario `theta` is shown at the top of the figure.

The results for each implemented control strategy versus the uncontrolled case are also captured using `theta`'s performance metric seen in Eq. (1). As the performance metric is ultimately a sum of penalties for violating the control objective, a smaller calculated performance metric value indicates a better performing control algorithm. The respective performance metric values for each control strategy presented here can be seen in Table 3. With a calculated performance metric of 0, the equal-filling degree strategy meets `theta`'s control objective; comparatively, the rule-based and uncontrolled cases have higher performance metric values, and thus, we can conclude perform worse than the equal-filling degree.

4.2. Additional demos

For demonstration purposes, Scenario `theta` was chosen due to its simplicity. For more complex examples, we direct the users to the Github repository where a Jupyter Notebook has been developed for each of the scenarios presented in Table 2, and a control strategy is implemented and evaluated against the uncontrolled case using the corresponding performance metric. The details for each scenario are provided in the A. In particular, we direct the reader to Scenarios `beta` (A.3), `gamma` (A.4), `epsilon` (A.6), and `zeta` (A.7), in which advanced controllers have been implemented, and the development of these controllers documented in previous research from Sadler et al. (2020), Mullapudi et al. (2020), Troutman et al. (2020), and Sun et al. (2020), respectively.

5. Discussion

The ability for stormwater systems to be rapidly modified is critical as communities prepare for climate change and more frequent, uncertain, and destructive weather events. Moreover, *often the most basic control strategies* can have large-scale impacts on the complex, dynamic systems they operate, potentially leading to millions of dollars in savings for the communities they serve. Even though sensor-actuator components may be successfully deployed at individual sites throughout a stormwater network, determining strategies for their coordination across the watershed may only add further complexity. As a result, there is a great need – along with numerous opportunities – to develop and implement novel control strategies to transform stormwater systems. The sandboxing efforts of `pystorms` serves as an initial effort to foster the development of these strategies. Moreover, we intend for `pystorms` to serve as a catalyst for our research community to be more expansive and comprehensive in its analysis of smart stormwater control.

A critical limitation to progressing smart stormwater control research forward is the inability to systematically develop and then analyze smart stormwater simulation workflows and control strategies. `pystorms` directly enables this development and analysis to become more extensive. `pystorms` can be customized and adapted for a multitude of applications beyond its initially-provided collection of scenarios, and additional research questions can be studied by assembling new scenarios from the assortment of components of this scenario collection. For example, for each of the scenarios we provide at the outset, `pystorms` specifies only a subset of the scenario's total observable states that are able to be queried throughout the simulation. But this initial subset of observable states is never claimed as the optimal; and actually, identifying an optimal set of observable states is in and of itself a rich area of research (Chacon-Hurtado et al., 2017; Sambito et al., 2019; Bartos and Kerkez, 2020; Van Nguyen et al., 2021). Thus, new scenarios can be made with different subsets of observable states (e.g. flows, pollutant concentrations), and new research questions can now be asked about which states may be most critical for informing control actions to be taken.

In the current iteration of `pystorms`, we present one means for assessing a control strategy via the scenario's corresponding performance metric (e.g. maintain flow below a threshold, avoid flooding). However, `pystorms` can serve as a mechanism for developing and implementing additional metrics. For example, by increasing the number of controllable assets available out of the eleven pond outlets presented in Scenario `gamma`, one can assess the *scalability* of a control algorithm as the state–action space increases. Additionally, control algorithm *generalizability* across storm characteristics can be assessed with the multiple rain events provided in Scenario `epsilon`.

Beyond the coordination and integration of smart stormwater control methods, we also view a more expansive – and potentially consequential – opportunity for `pystorms` to drive our research community's analysis of "control" to include the social and ethical implications of its implementation. We recognize that the control of stormwater

Table 3
Calculated performance metric values from Eq. (1) for simulations corresponding to the two implemented control algorithms and the uncontrolled simulation. As can be seen, the equal-filling degree control strategy performs better than the rule-based control strategy, which then outperforms the uncontrolled case.

Control strategy	Performance metric
Uncontrolled	1630
Rule-based	1624
Equal-filling degree	0

systems operates within broader, and far more complex, socio-technical systems, and as a result, the decisions made can have a profound impact on the lives of people within those systems. Thus, in addition to water quantity and quality analyses, we impel our community to also scrutinize their control strategies through the lens of social equity and longterm community resilience and adaptiveness, such as the decision framework developed by Ewing and Demir (2021). We believe `pystorms` can help facilitate these analyses. For example, for any of the provided scenarios, one could study the longterm social implications of an adverse outcome that might occur regularly in the same community due to an implemented control strategy by modifying a scenario's control objective and corresponding performance metric.

Finally, we encourage future users of `pystorms` to resist the natural inclination towards fixating on a control algorithm's *performance* at the expense of its *appropriateness*. Our development of `pystorms` was partly motivated by similar sets of data and libraries developed by the electrical and computer science communities over the past two decades in an effort to evaluate their ever-growing assortment of machine-learning algorithms (Deng, 2012; Russakovsky et al., 2015; Brockman et al., 2016; Henderson et al., 2017). While their efforts have been wildly successful at propelling the development of novel algorithms, it has also been noted that much of that effort has been expended in the "fine-tuning" and "hacking" of the various algorithms to achieve some sort of arbitrarily-defined "state-of-the-art" result (Torralba and Efros, 2011). While fine-tuning an algorithm's behavior for a specific dataset may indeed yield a better performance metric in `pystorms`, this improved performance metric is not guaranteed to translate to the physical system at hand. Validating these stormwater control strategies requires much further analysis via their actual physical deployment. Accordingly, we intend for `pystorms` to foster a collaborative and thoughtful evaluation of stormwater control algorithms rather than a myopic focus on performance metrics and competition.

6. Conclusions and next steps

`pystorms` provides a curated collection of scenarios, coupled with an accessible programming interface, to enable the development and quantitative evaluation of stormwater control algorithms. We have developed `pystorms` with the intent to make smart stormwater control research more methodical and efficient. As shown with the demos in Section 4 and accompanying tutorials, we have demonstrated how users can quickly download and test `pystorms` and its scenarios on a basic computer with only a few lines of code. Additionally shown in Section 4, `pystorms` facilitates rigorous evaluation by extricating the control algorithm implementation of a stormwater control strategy to be applied and quantitatively compared across the example scenarios.

It is our hope that this package will emerge as a community-driven resource that is able to address key knowledge gaps and enable the advancement of smart stormwater systems. To this extent, we see proximate opportunities for the broader research community to collaborate on `pystorms` by contributing their own stormwater scenarios and/or control algorithms to the package initiated here. Likewise, we encourage the broader research community to further build upon `pystorms` by imparting their own smart stormwater control instances using the `pystorms` architecture and integrating their own stormwater control simulation workflows into it. Furthermore, given the emergence of new stormwater practices, such as Green Infrastructure, more research is needed to model impacts of real-time control on these assets with sufficient fidelity.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:Branko Kerkez reports financial support, administrative support, and equipment, drugs, or supplies were provided by National Science Foundation.

Data availability

All data and code have been made available on a public repository

Acknowledgments

This research was supported U.S. National Science Foundation, Award Numbers: 1737432 and 1750744. Additionally, Argonne National Laboratory's contribution is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357. We would also like to acknowledge the Great Lakes Water Authority for their support.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.envsoft.2023.105635.

References

Advanced Research Projects Agency–Energy, U.S. Department of Energy, 2016. Generating Realistic Information for the Development of Distribution and Transmission Algorithms (GRID DATA) Program. https://arpa-e.energy.gov/technologies/programs/grid-data.

Bartos, Matt, Kerkez, Branko, 2020. Observability-based sensor placement improves contaminant tracing in river networks. Earth Space Sci. Open Arch. http://dx.doi.org/10.1002/essoar.10504108.1.

Batty, Michael, Axhausen, Kay W, Giannotti, Fosca, Pozdnoukhov, Alexei, Bazzani, Armando, Wachowicz, Monica, Ouzounis, Georgios, Portugali, Yuval, 2012. Smart cities of the future. Eur. Phys. J. Spec. Top. 214 (1), 481–518. http://dx.doi.org/10.1140/epjst/e2012-01703-3.

Bibri, Simon Elias, Krogstie, John, 2017. Smart sustainable cities of the future: An extensive interdisciplinary literature review. Sustainable Cities Soc. 31, 183–212. http://dx.doi.org/10.1016/j.scs.2017.02.016.

Borsányi, Péter, Benedetti, Lorenzo, Dirckx, Geert, De Keyser, Webbey, Muschalla, Dirk, Solvi, Anne-Marie, Vandenberghe, Veronique, Weyand, Michael, Vanrolleghem, Peter A., 2008. Modelling real-time control options on virtual sewer systems. J. Environ. Eng. Sci. 7 (4), 395–410. http://dx.doi.org/10.1139/S08-004.

Bowes, Benjamin D, Tavakoli, Arash, Wang, Cheng, Heydarian, Arsalan, Behl, Madhur, Beling, Peter A, Goodall, Jonathan L, 2021. Flood mitigation in coastal urban catchments using real-time stormwater infrastructure control and reinforcement learning. J. Hydroinform. 23 (3), 529–547. http://dx.doi.org/10.2166/hydro.2020.080.

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, Zaremba, Wojciech, 2016. OpenAI Gym. arXiv:arXiv:1606.01540.

Campisano, Alberto, Schilling, Wolfgang, Modica, Carlo, 2000. Regulators' setup with application to the Roma–Cecchignola combined sewer system. Urban Water 2 (3), 235–242. http://dx.doi.org/10.1016/S1462-0758(00)00061-3.

Chacon-Hurtado, J.C., Alfonso, L., Solomatine, D.P., 2017. Rainfall and streamflow sensor network design: A review of applications, classification, and a proposed framework. Hydrol. Earth Syst. Sci. 21 (6), 3071–3091. http://dx.doi.org/10.5194/hess-21-3071-2017.

Chourabi, Hafedh, Nam, Taewoo, Walker, Shawn, Gil-Garcia, J. Ramon, Mellouli, Sehl, Nahon, Karine, Pardo, Theresa A., Scholl, Hans Jochen, 2012. Understanding smart cities: An integrative framework. In: Proceedings of the Annual Hawaii International Conference on System Sciences. IEEE, pp. 2289–2297. http://dx.doi.org/10.1109/HICSS.2012.615.

Deng, Li, 2012. The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. 29 (6), 141–142.

Dirckx, G., Schütze, M., Kroll, S., Thoeye, Ch., De Gueldre, G., Van De Steene, B., 2011. Cost-efficiency of RTC for CSO impact mitigation. Urban Water J. 8 (6), 367–377. http://dx.doi.org/10.1080/1573062X.2011.630092.

Eggimann, Sven, Mutzner, Lena, Wani, Omar, Schneider, Mariane Yvonne, Spuhler, Dorothee, Moy de Vitry, Matthew, Beutler, Philipp, Maurer, Max, 2017. The potential of knowing more: A review of data-driven urban water management. Environ. Sci. Technol. 51 (5), 2538–2553. http://dx.doi.org/10.1021/acs.est.6b04267.

Ewing, Gregory, Demir, Ibrahim, 2021. An ethical decision-making framework with serious gaming: A smart water case study on flooding. J. Hydroinform. http://dx.doi.org/10.2166/hydro.2021.097.

Gaborit, E., Anctil, François, Pelletier, Geneviève, Vanrolleghem, Peter A., 2016. Exploring forecast-based management strategies for stormwater detention ponds. Urban Water J. 13 (8), http://dx.doi.org/10.1080/1573062X.2015.1057172.

Gaborit, E., Muschalla, Dirk, Vallet, Bertrand, Vanrolleghem, Peter A., Anctil, François, 2013. Improving the performance of stormwater detention basins by real-time control using rainfall forecasts. Urban Water J. 10 (4), 230–246. http://dx.doi.org/10.1080/1573062X.2012.726229.

García, L., Barreiro-Gomez, J., Escobar, E., Téllez, D., Quijano, N., Ocampo-Martinez, C., 2015. Modeling and real-time control of urban drainage systems: A review. Adv. Water Resour. 85, 120–132. http://dx.doi.org/10.1016/j.advwatres.2015.08.007.

Harrison, Colin, Donnelly, Ian Abbott, 2011. A theory of smart cities. In: Proceedings of the 55th Annual Meeting of the International Society for the Systems Sciences, Vol. 55, no. 1.

Henderson, Peter, Islam, Riashat, Bachman, Philip, Pineau, Joelle, Precup, Doina, Meger, David, 2017. Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560.

Kerkez, Branko, Gruden, Cyndee, Lewis, Matthew, Montestruque, Luis, Quigley, Marcus, Wong, Brandon, Bedig, Alex, Kertesz, Ruben, Braun, Tim, Cadwalader, Owen, Poresky, Aaron, Pak, Carrie, 2016. Smarter stormwater systems. Environ. Sci. Technol. 50 (14), 7267–7273. http://dx.doi.org/10.1021/acs.est.5b05870.

Kitchin, Rob, 2014. The real-time city? Big data and smart urbanism. GeoJournal 79 (1), 1–14. http://dx.doi.org/10.1007/s10708-013-9516-8.

Kroll, Stefan, Dirckx, Geert, Donckels, Brecht M.R., Van Dorpe, Mieke, Weemaes, Marjoleine, Willems, Patrick, 2016. Modelling real-time control of WWTP influent flow under data scarcity. Water Sci. Technol. 73 (7), 1637–1643. http://dx.doi.org/10.2166/wst.2015.641.

Lund, Nadia Schou Vorndran, Falk, Anne Katrine Vinther, Borup, Morten, Madsen, Henrik, Steen Mikkelsen, Peter, 2018. Model predictive control of urban drainage systems: A review and perspective towards smart real-time water management. Crit. Rev. Environ. Sci. Technol. 48 (3), 279–339.

Marchi, Angela, Salomons, Elad, Ostfeld, Avi, Kapelan, Zoran, Simpson, Angus R., Zecchin, Aaron C., Maier, Holger R., Wu, Zheng Yi, Elsayed, Samir M., Song, Yuan, Walski, Tom, Stokes, Christopher, Wu, Wenyan, Dandy, Graeme C., Alvisi, Stefano, Creaco, Enrico, Franchini, Marco, Saldarriaga, Juan, Páez, Diego, Hernández, David, Bohórquez, Jessica, Bent, Russell, Coffrin, Carleton, Judi, David, McPherson, Tim, van Hentenryck, Pascal, Matos, José Pedro, Monteiro, António Jorge, Matias, Natércia, Yoo, Do Guen, Lee, Ho Min, Kim, Joong Hoon, Iglesias-Rey, Pedro L., Martínez-Solano, Francisco J., Mora-Meliá, Daniel, Ribelles-Aguilar, José V., Guidolin, Michele, Fu, Guangtao, Reed, Patrick, Wang, Qi, Liu, Haixing, McClymont, Kent, Johns, Matthew, Keedwell, Edward, Kandiah, Venu, Jasper, Micah Nathanael, Drake, Kristen, Shafiee, Ehsan, Barandouzi, Mehdy Amirkhanzadeh, Berglund, Andrew David, Brill, Downey, Mahinthakumar, Gnanamanikam, Ranjithan, Ranji, Zechman, Emily Michelle, Morley, Mark S., Tricarico, Carla, de Marinis, Giovanni, Tolson, Bryan A., Khedr, Ayman, Asadzadeh, Masoud, 2014. Battle of the water networks II. J. Water Res. Plan. Manag. 140 (7), 04014009. http://dx.doi.org/10.1061/(ASCE)WR.1943-5452.0000378.

McDonnell, Bryant E, Ratliff, Katherine, Tryby, Michael E, Wu, Jennifer Jia Xin, Mullapudi, Abhiram, 2020. PySWMM: The Python interface to stormwater management model (SWMM). J. Open Source Softw. 5 (52), 2292.

Montestruque, Luis A., 2018. An agent-based storm water management system. In: Tsakalides, Panagiotis, Panousopoulou, Athanasia, Tsagkatakis, Grigorios, Montestruque, Luis (Eds.), Smart Water Grids: A Cyber-Physical Systems Approach. CRC Press, pp. 151–168.

Mullapudi, Abhiram, Lewis, Matthew J, Gruden, Cyndee L, Kerkez, Branko, 2020. Deep reinforcement learning for the real time control of stormwater systems. Adv. Water Resour. 140, 103600.

Mullapudi, Abhiram, Wong, Brandon P., Kerkez, Branko, 2017. Emerging investigators series: Building a theory for smart stormwater systems. Environ. Sci. Water Res. Technol. 3 (1), http://dx.doi.org/10.1039/C6EW00211K.

Newman, A.J., Clark, M.P., Sampson, K., Wood, A., Hay, L.E., Bock, A., Viger, R.J., Blodgett, D., Brekke, L., Arnold, J.R., Hopson, T., Duan, Q., 2015. Development of a large-sample watershed-scale hydrometeorological data set for the contiguous USA: Data set characteristics and assessment of regional variability in hydrologic model performance. Hydrol. Earth Syst. Sci. 19 (1), 209–223. http://dx.doi.org/10.5194/hess-19-209-2015.

Ocampo-Martinez, Carlos, 2010. Model predictive control of wastewater systems. Springer, pp. 1–236. http://dx.doi.org/10.1007/978-1-84996-353-4.

Ostfeld, Avi, Uber, James G., Salomons, Elad, Berry, Jonathan W., Hart, William E., Phillips, Cindy A., Watson, Jean-Paul, Dorini, Gianluca, Jonkergouw, Philip, Kapelan, Zoran, di Pierro, Francesco, Khu, Soon-Thiam, Savic, Dragan, Eliades, Demetrios, Polycarpou, Marios, Ghimire, Santosh R., Barkdoll, Brian D., Gueli, Roberto, Huang, Jinhui J., McBean, Edward A., James, William, Krause, Andreas, Leskovec, Jure, Isovitsch, Shannon, Xu, Jianhua, Guestrin, Carlos, VanBriesen, Jeanne, Small, Mitchell, Fischbeck, Paul, Preis, Ami, Propato, Marco, Piller, Olivier, Trachtman, Gary B., Wu, Zheng Yi, Walski, Tom, 2008. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. J. Water Res. Plan. Manag. 134 (6), 556–568. http://dx.doi.org/10.1061/(ASCE)0733-9496(2008)134:6(556).

Persaud, P.P., Akin, A.A., Kerkez, B., McCarthy, D.T., Hathaway, J.M., 2019. Real time control schemes for improving water quality from bioretention cells. Blue-Green Syst. 1 (1), 55–71. http://dx.doi.org/10.2166/bgs.2019.924.

Rossman, L., 2015. Storm Water Management Model User's Manual Version 5.1 - Manual, EPA/600/R-14/413 (NTIS EPA/600/R-14/413b) ed. US EPA Office of Research and Development, Washington, DC.

Rossman, L., 2017. Storm Water Management Model Reference Manual Volume II – Hydraulics, EPA/600/R-17/111 ed. US EPA Office of Research and Development, Washington, DC.

Rossman, L., Huber, W., 2015. Storm Water Management Model Reference Manual Volume I – Hydrology, EPA/600/R-15/162A ed. US EPA Office of Research and Development, Washington, DC.

Rossman, L., Huber, W., 2016. Storm Water Management Model Reference Manual Volume III – Water Quality, EPA/600/R-16/093 ed. US EPA Office of Research and Development, Washington, DC.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al., 2015. ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. 115 (3), 211–252. http://dx.doi.org/10.1007/s11263-015-0816-y.

Sadler, Jeffrey M., Goodall, Jonathan L., Behl, Madhur, Bowes, Benjamin D., Morsy, Mohamed M., 2020. Exploring Real-time Control of Stormwater Systems for Sea Level Rise. J. Hydrol. http://dx.doi.org/10.1016/j.jhydrol.2020.124571.

Sadler, Jeffrey M., Goodall, Jonathan L., Behl, Madhur, Morsy, Mohamed M., Culver, Teresa, Bowes, Benjamin D., 2019. Leveraging open source software and parallel computing for model predictive control of urban drainage systems using EPA-SWMM5. Environ. Model. Softw. http://dx.doi.org/10.1016/j.envsoft.2019.07.009.

Sambito, Mariacrocetta, Di Cristo, Cristiana, Freni, Gabriele, Leopardi, Angelo, 2019. Optimal water quality sensor positioning in urban drainage systems for illicit intrusion identification. J. Hydroinform. 22 (1), 46–60. http://dx.doi.org/10.2166/hydro.2019.036.

Schilling, Wolfgang, 1989. Real-time control of urban drainage systems: The state-of-the-art. IAWPRC Task Group on Real-Time Control of Urban Drainage Systems, London.

Schütze, Manfred, Campisano, Alberto, Colas, Hubert, Schilling, Wolfgang, Vanrolleghem, Peter A., 2004. Real time control of urban wastewater systems - Where do we stand today? J. Hydrol. 299 (3–4), 335–348. http://dx.doi.org/10.1016/j.jhydrol.2004.08.010.

Schütze, Manfred, Lange, Maja, Pabst, Michael, Haas, Ulrich, 2017. Astlingen - a benchmark for real time control (RTC). Water Sci. Technol. 2017 (2), 552–560.

Shishegar, Shadab, Duchesne, Sophie, Pelletier, Geneviève, 2018. Optimization methods applied to stormwater management problems: A review. Urban Water J. 15 (3), 276–286.

Sun, Congcong, Svensen, Jan Lorenz, Borup, Morten, Puig, Vicenç, Cembrano, Gabriela, Vezzaro, Luca, 2020. An MPC-enabled SWMM implementation of the Astlingen RTC benchmarking network. Water 12 (1034), 1–13. http://dx.doi.org/10.3390/w12041034.

Torralba, Antonio, Efros, Alexei A., 2011. Unbiased look at dataset bias. In: CVPR 2011. IEEE, pp. 1521–1528.

Trotta, Paul D., Labadie, J.W., Grigg, N.S., 1977. Automatic control strategies for urban stormwater. J. Hydraul. Div. 1443–1459.

Troutman, Sara C., Love, Nancy G., Kerkez, Branko, 2020. Balancing water quality and flows in combined sewer systems using real-time control. Environ. Sci. Water Res. Technol. 6 (5), 1357–1369.

van Daal, Petra, Gruber, Günter, Langeveld, Jeroen, Muschalla, Dirk, Clemens, François, 2017. Performance evaluation of real time control in urban wastewater systems in practice: Review and perspective. Environ. Model. Softw. 95, 90–101.

Van Nguyen, Lam, Bui, Dieu Tien, Seidu, Razak, 2021. Identification of Sensitive Factors for Placement of Flood Monitoring Sensors in Wastewater/Stormwater Network Using GIS-Based Fuzzy Analytical Hierarchy Process: A Case of Study in Ålesund, Norway. In: Tien Bui, Dieu, Tran, Hai Thanh, Bui, Xuan-Nam (Eds.), Proceedings of the International Conference on Innovations for Sustainable and Responsible Mining. Springer International Publishing, pp. 79–97.

Vanrolleghem, Peter A., Benedetti, Lorenzo, Meirlaen, J., 2005. Modelling and real-time control of the integrated urban wastewater system. Environ. Model. Softw. 20 (4), 427–442. http://dx.doi.org/10.1016/j.envsoft.2004.02.004.

Vezzaro, Luca, Grum, Morten, 2014. A generalised Dynamic Overflow Risk Assessment (DORA) for real time control of urban drainage systems. J. Hydrol. 515, 292–303. http://dx.doi.org/10.1016/j.jhydrol.2014.05.019.

Walski, Thomas M., Brill, E. Downey, Gessler, Johannes, Goulter, Ian C., Jeppson, Roland M., Lansey, Kevin, Lee, Han-Lin, Liebman, Jon C., Mays, Larry, Morgan, David R., Ormsbee, Lindell, 1987. Battle of the network models: Epilogue. J. Water Res. Plan. Manag. 113 (2), 191–203. http://dx.doi.org/10.1061/(ASCE)0733-9496(1987)113:2(191).

Yuan, Zhiguo, Olsson, Gustaf, Cardell-Oliver, Rachel, van Schagen, Kim, Marchi, Angela, Deletic, Ana, Urich, Christian, Rauch, Wolfgang, Liu, Yanchen, Jiang, Guangming, 2019. Sweating the assets - The role of instrumentation, control and automation in urban water systems. Water Res. 155, 381–402. http://dx.doi.org/10.1016/j.watres.2019.02.034.