Modeling Fluid Flow in RDDL for Simulation of Flow Control Systems

Donovan Bisson[1]

School of Computing, Queen's University, Kingston, Canada

Abstract
Coming soon...

1. Introduction

In process control systems, the full state space is typically known but only partially observed. Important values are monitored by sensors in pertinent locations to give a sense of the system at large, but availability of sensor data for a system tends to become more sparse as the system expands. State estimation can fill knowledge gaps by interpolating how the system would function in given conditions. However, such state estimation methods are domain-specific; they need a model that can represent features of the system and simulate the underlying rules of the system.
With respect to control systems, such models are called model-predictive control (MPC) systems, and they are extremely useful for non-linear control problems. One such example would be flow control problems - moving fluid through a system of pipes and reservoirs. These kinds of problems include water treatment and distribution, wastewater treatment, and chemical industry processes. Solving these problems often requires a robust modeling scheme, both for simulating the system to build a policy and for estimating action effects in real time.
The goal of this model is to act as a realistic domain representation of flow control systems, one that can both act as a simulator for training an offline policy and a way to estimate states for an online policy.

2. Background

2.1. Overview - Model Predictive Control

https://link.springer.com/article/10.1007/s00170-021-07682-3
Robust analysis of model-predictive control. Key concepts:

- MPC systems model a state representation of a system, which is used to predict the effects of actions within the system. This turns control problems into constrained temporal optimization problems (great for planning!)
- MPC can control non-linear models that traditional control systems struggle with. Very valuable in process industries.
- Broadly speaking, this article is here to define MPC and to justify use of a planning model for process-based problem spaces such as flow control. Nothing specifically RDDL related, but MPC systems have found use all over the place from my reading - I'll specify some examples in the final report.

2.2. Flow Networks

https://dspace.mit.edu/bitstream/handle/1721.1/49424/networkflows00ahuj.pdf
Outlines how flow networks are defined, as well as common problems they can be used to solve (max flow, min-cost flow, etc.).

2.3. Types of Pumps, Formulas, Pump Curves

- Positive-displacement pumps - move fixed volumes of fluid per cycle, volumetric flow mostly independent of pressure (above threshold, called "Net Positive Suction Head")
- Centrifugal pumps - use impellers to generate flow via kinetic energy, ratio of cycle rate:volumetric flow becomes a function of fluid pressure.
- Pump curves - Represented as a curve, but more of a topology. Pressure on one axis, flow rate on the other, mapping speed of the pump forms a surface in 3D space. This is really important, as systems that use pump speed as the control variable and flow rate/pressure as the process variable need to compensate for changes in one or the other due to environmental factors (turbulence) and upstream/downstream changes.

3. An RDDL Planning Model for Continuous Fluid Control Systems

3.1. Types

3.1.1. Nodes

Nodes in the system are synonymous with any fluid storage device that does not compress the fluid, such as tanks, reservoirs, lakes or other containers. The main assumption made with nodes is that the area at any elevation within node will be equal to any other elevation within the node (though other assumptions will be made with respect to nodes shortly).
Nodes store fluid up to a finite capacity (defined in the problem file) and are connected to each other by directed arcs, where fluid moves from one node to another. The rate of flow is controlled by pumps, which are described below.

3.1.2. Pumps

Positive-Displacement (PD)
Positive-displacement pumps function by moving a constant volume of fluid per revolution, and will always produce the necessary pressure to move the fluid. This makes them relatively simple to model, as their flow output is independent of head pressure. As a result, PD pumps function as deterministic flow controllers due to stochasticity in the model being only introduced via variation in head pressure.

Centrifugal (CF)
Centrifugal pumps are unlike PD pumps in that they induce flow using an impeller instead of collecting and moving a constant amount of fluid at a time. Since the necessary pressure that must be induced by the pump to reach a certain flow is dependent on the total dynamic head, the model must account for both pump speed and total dynamic head when calculating flow.
Calculating TDH depends on a multitude of factors, and as such assumptions had to be made to simplify the model for sake of usability. Specifically, we assume that i. Pumps draw from the bottom of each tank, ii. Tanks are situated at equal elevation from each other, iii. There is enough fluid between the inlet/outlet of the tanks and the pumps that the minimum necessary suction head is always met so long as the tanks are not empty, and iv. The system operates under standard gravity (9.8 m/s^2) and atmospheric pressure (14.7 psi) at the surface of fluid within the tanks. With these assumptions, the formula for TDH can be simplified to the following:
$TDH = \text{Hydrostatic pressure downstream} - \text{Hydrostatic pressure upstream}$

... where the hydrostatic pressure imparted by connected tanks upstream/downstream is done via the following formula:
$\text{Hydrostatic pressure} = \text{Fluid density} * \text{Fluid depth} * 9.81 + 14.7$
The relationship between flow and head pressure is approximated by the Pump Affinity Laws as inversely proportional to the square of the head pressure, so the flow calculations subtract the square of the head pressure from the formula used for PD flow.
Note that CF pumps are the only source of stochasticity in the model, as any variation in head pressure will affect CF pump output but not PD pump output.

3.2. Non-fluents

3.2.1. Constants

```
POLLING-RATE
  type: real   default: 1.0
```
The time between state snapshots (s). Faster polling rates increase system responsiveness, but limit the time the system has to build plans.

```
FLUID-DENSITY
  type: real   default: 1000.0
```
The density of the fluid in the system (kg/m^3).

```
MAX-FLOW-REWARD
  type: real   default: 100.0
```
The reward for reaching a flow target.

```
MAX-LEVEL-REWARD
  type: real   default: 100.0
```
The reward for reaching a level target.

```
FLOW-REWARD-DROPOFF
  type: real   default: 1.0
```
How much each unit of deviation from a flow target is penalized.

```
LEVEL-REWARD-DROPOFF
  type: real   default: 1.0
```
How much each unit of deviation (as % of capacity) from a level target is penalized.

3.2.2. System Attributes

```
SOURCE(node)
  type: bool   default: false
```
Source nodes are points where fluid enters/exits the system. The presence of a source node differentiates a closed system from an open system.

`DRAWS(pump, node, node)`
  `type: bool   default: false`
Directed arcs in the system graph, where the pump draws fluid from the first node to the second node.

3.2.3. Object Attributes

`CAPACITY(node)`
  `type: real   default: 1000.0`
The maximum holding capacity of a node (m^3). Must be greater than zero.

`MAX-DEPTH(node)`
  `type: real   default: 20.0`
The height of the node (m). Necessary for calculating pressure and pump head.

`PUMP-TYPE(pump)`
  `type: pump-type   default: @pd`
The type of a given pump. Can be either positive-displacement or centrifugal.

`PUMP-DISPLACEMENT(pump)`
  `type: real   default: 1.0`
The amount of fluid moved by a single cycle of a given pump (m^3/cycle).

`HAS-TARGET-FLOW(pump)`
  `type: bool   default: false`
Whether a given pump has a flow target.

`HAS-TARGET-LEVEL(node)`
  `type: bool   default: false`
Whether a given node has a level target.

`TARGET-FLOW(pump)`
  `type: real   default: 0.0`
The target flow value for a pump, if such a target exists (m^3/s).

`TARGET-LEVEL(node)`
  `type: real   default: 0.0`
The target level for a node, if such a target exists (% of capacity).

3.3. Intermediate Fluents

`tank-level(node)`
  `type: real`
The current level of a tank (% of capacity)

`tank-pressure(node)`
  `type: real`
The current hydrostatic pressure applied by a tank to any pumps connected to it (psi). Based on the standard formula for hydrostatic pressure. It is assumed that pumps are connected to the bottom of each tank, and uses standard values for gravity (9.81 m/s^2) and surface pressure (1 atm).

`pump-expected-head(pump)`
  `type: real`
The expected total dynamic head (TDH) of the pump, given the head pressures of connected tanks. Assumes the minimum required Net Positive Suction Head (NPSHr) is always met, and that all other factors are equal. Converts from psi to TDH using the standard conversion factor of 2.31.

`pump-extra-head(pump)`
  `type: real`
Any additional head pressure experienced by the pump beyond what would be expected given the head pressures of connected tanks (such as from turbulence or obstruction). Introduces stochasticity to any system with centrifugal pumps (PD pumps are pressure-invariant). Uses RDDL's built-in Beta probability distribution.

`pump-head(pump)`
  `type: real`
The total head pressure experienced by the pump, after introducing stochasticity.

`new-pump-hz(pump)`
  `type: real`
The new speed of the pump, after accelerating the pump from the current speed by `pump-delta(pump)`.

`new-pump-flow(pump)`
  `type: real`
The flow rate of the pump at the current head pressure, when the pump is operating at `new-pump-hz(pump)`. Varies by pump type:

- `@pd`: Linearly proportional to the pump speed regardless of head pressure.
- `@cf`: Linearly proportional to the pump speed, AND inversely proportional to the square of the head pressure.

`inflow(node)`
  `type: real`
The rate that fluid is entering a tank, based on the new flow rates of pumps drawing into the tank.

```
outflow(node)
  type: real
```
The rate that fluid is exiting a tank, based on the new flow rates of pumps drawing from the tank.

3.4. State and Action Fluents

3.4.1. State Fluents

```
pump-hz(pump)
  type: real   default: 0.0
```
The speed of the pump (Hz) in the given state. Updates by adding any change in pump speed to the current speed. Since the model controls the change in pump speeds, the speed at the current state affects how much the model can change the speed. Thus, knowledge of current speed must be retained between states.
Subject to the following state invariants:

- The speed of the pump must be a non-negative real number.

```
pump-flow(pump)
  type: real   default: 0.0
```
The current flow rate of the pump (m^3/s) in the given state. Added for purposes of visualization, does not contribute to fluent calculations.
Subject to the following state invariants:

- The flow rate of the pump must be a non-negative real number.

```
holding(node)
  type: real   default: 0.0
```
The total volume of fluid stored in a tank (m^3). Updates by adding the difference between inflow and outflow to the tank, multiplied by the polling rate. The amount of stored fluid affects the allowable in/outflow to the tank and the head pressures of connected nodes, and thus must be retained when changing states.
Subject to the following state invariants:

- The tank must hold a non-negative amount of fluid, and
- The tank must not hold more than its maximum capacity.

```
cur-pump-delta(pump)
  type: real   default: 0.0
```
The acceleration of the pump in the current state (Hz/POLLING-RATE). Retained across states to let the reward function penalize large changes in acceleration ("jolt"), and for visualization.

3.4.2. Action Fluents

```
pump-delta(pump)
  type: real   default: 0.0
```
How much the model will increase/decrease the speed of the pump.
Subject to the following action constraints:

- The absolute acceleration is capped at 1 hz/s, scaling with polling rate;
- The absolute change in acceleration ("jolt") between the current and next state is capped at 0.5 hz/s^2, scaling with polling rate; and
- The change in speed cannot bring the pump speed below zero.

3.5. Reward Function

The reward function has been designed to incentivize maintaining any target values while avoiding "fail states" where state invariants are broken. It can be broken down into two components:

1. Target incentives, which reward meeting target values and penalize excessive deviation; and
2. Constraint enforcements, which heavily penalize the model for reaching a fail state.

Calculating the reward from a given state-action pair is to combine the values of each component. Each component is explained in detail below.

3.5.1. Target Incentives

To incentivize the model to approach and maintain target flow/level values, each target is assigned two values:

1. A maximum reward value, given when the state value is equal to the target value; and
2. A decay value, which is deducted from the total reward for each unit (either m^3/s or % capacity) of deviation between the state and target value.

Decay linearly deducts reward from the total based on deviation, maintaining a continuous and measurable reward gradient to improve convergence in the planner. It is important to note that poor assignment of these values (specifically large negative max rewards and overly large decay values) can lead to the system intentionally breaking state constraints to maximize reward. Additionally, the values should be manually adjusted based on the number of targets in the system for the same reason, as each target adds to the reward independently.

3.5.2. Constraint Enforcement

To dissuade the model from premature termination by breaking state constraints, large negative rewards are given for any action that would cause the system to fail. A large static negative reward is applied to any applicable action, as well as an additional penalty that scales linearly based on the distance from the allowable state space. The static penalty breaks continuity in the reward space to make moving outside the state space from within nearly impossible (excepting some edge cases), and the linear penalty allows the planning agent to quickly learn to escape the invalid action space.

4. Evaluation

4.1. Problem Instances

Online linear replanning with JaxStraightLinePlan `(rollout_horizon=5, train_seconds=5, utility='cvar', utility_kwargs = {'alpha': 0.2})` was applied to five unique problem instances (connection diagrams in 1) `(POLLING-RATE = 5)`:

1. instance0 - Simple closed system
   Targets:
   - p1: 50 m^3/s
   - t1: 50% capacity

   Goal: Evaluate systemic conservation of fluid.
2. instance1 - Simple open system
   Targets:
   - p2: 50 m^3/s
   - t2: 50% capacity

   Goal: Compare dynamics of closed vs open systems.
3. instance11 - Simple open system with distributed flow
   Targets:
   - p2: 50 m^3/s
   - t2: 50% capacity

   Goal: Observe behavior when flow can be split between multiple pumps.
4. instance2 - Complex open system for flow control
   Targets:
   - p1-1: 400 m^3/s
   - p2-1: 200 m^3/s
   - p3: 600 m^3/s

   Goals: Evaluate ability for system to stabilize flow rates in complex systems; compare ability to meet PD vs CF pump targets.
5. instance3 - Complex open system for level control
   Targets:
   - t2-1: 80% capacity
   - t2-2: 40% capacity
   - t3: 80% capacity
   - p3: 600 m^3/s

   Goal: Evaluate ability for system to stabilize tank levels in complex systems compared to flow rates.

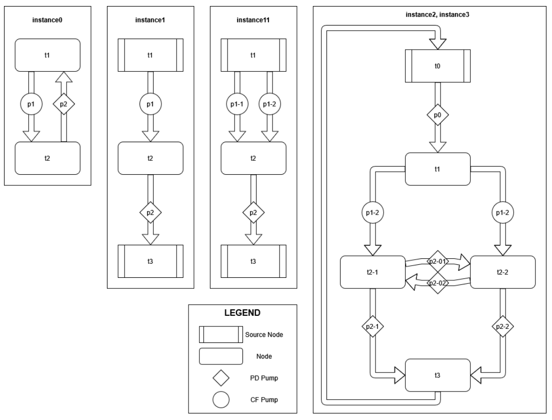

Figure 1: Diagrams of pump-node connectivity for each problem instance.

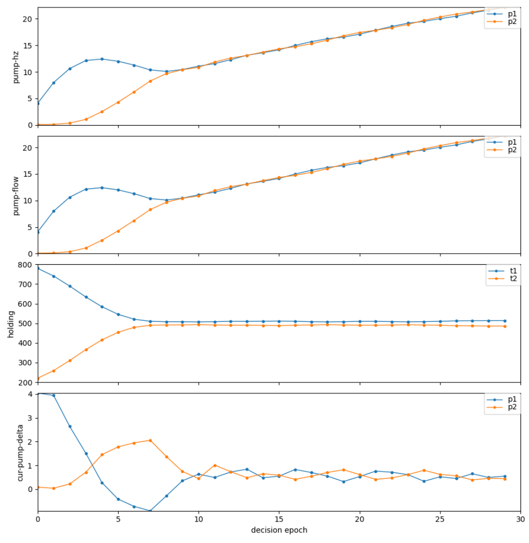

Figure 2: States of instance0 after 30 decision epochs

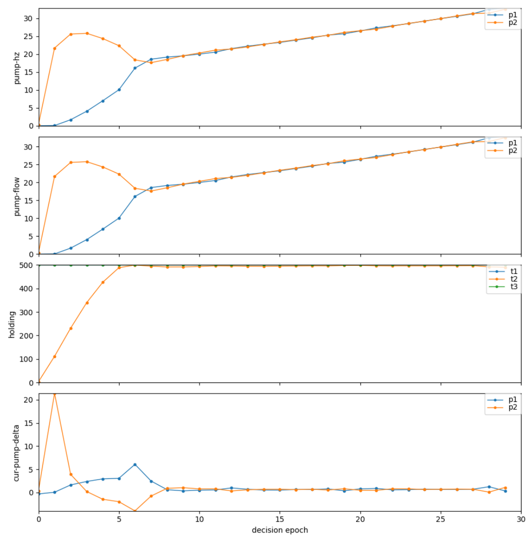

Figure 3: States of instance1 after 30 decision epochs

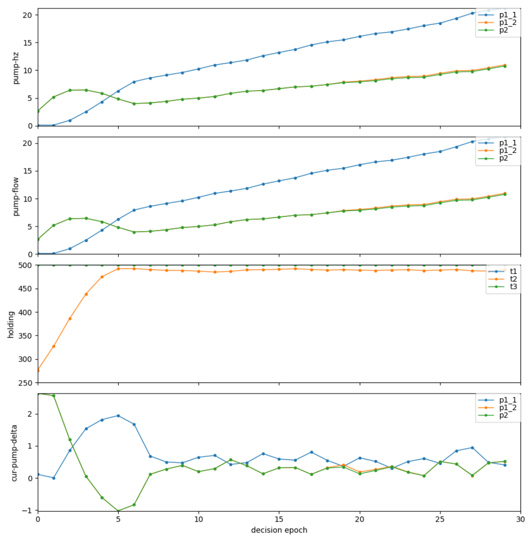

Figure 4: States of instance11 after 30 decision epochs

4.2. Results

4.2.1. instance0

See 2.
Observations:

- No fluid loss! Fluid held in either tank are symmetric about 50% of the fluid in the system at initialization.
- System quickly stabilized levels, but never met flow target. Reward function penalizing large deltas may be the reason, and/or the limited rollout window.

4.2.2. instance1

See 3.
Observations:

- Pump deltas closely match instance0 in shape, but with much greater magnitude in the first few epochs. May be due to being an open system, as source nodes don't risk overdrawing/overfilling.
- Same trend with matching flow vs level as in instance0.

4.2.3. instance11

See 4.
Observations:

- Similar trends to instance1, but magnitude of parallel pumps much smaller than before (more than the expected 50% reduction)! More CF pumps means more variance, so smaller changes are safer most of the time. Prioritizing safety is good, but may inhibit system convergence speed.
- Parallel pump speeds/deltas are almost identical in shape. The reward function penalizes the square of the deltas, so changing each by the same factor may be better than one by double the amount. Some variance is expected due to stochastic factor introduced by variable head pressure.

4.2.4. instance2

See 5.
Observations:

- Remarkably stable. Rewarding smaller changes seems to have the desired effect in preventing overcompensation.
- Despite only having flow targets, targeted pumps failed to meet the desired thresholds. May be result of over-punishing large deltas + small horizon as before.

4.2.5. instance3

See 6.
Observations:

- Despite the network being identical to that of instance2 beyond targets, instance3 sees much larger deltas and much faster convergence.

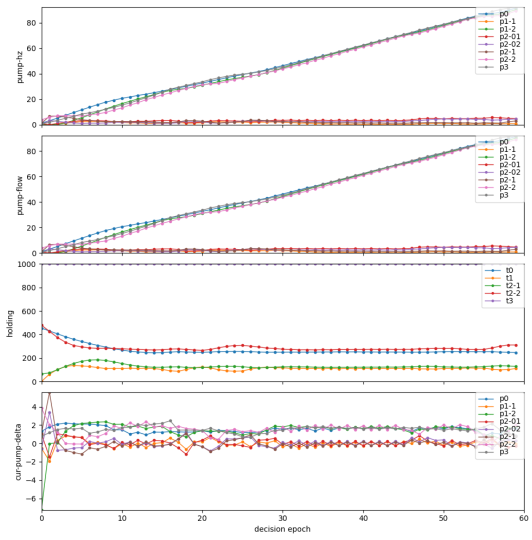

Figure 5: States of instance2 after 60 decision epochs

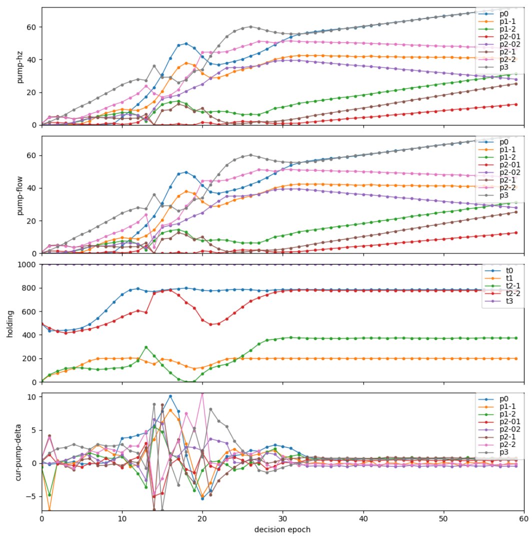

Figure 6: States of instance3 after 60 decision epochs

- Level targets seem to have a much greater effect on the model than flow targets with the current configuration.
- Pumps continue to exhibit changes in speed beyond adjusting for variance; the single flow target was also not met by simulation end. Again, the implication is an issue with the reward function overpenalizing large deltas - changing the magnitude of the penalty may benefit the model.

5. Related Work

5.1. IPPC - Reservoir (cont.)

https://github.com/pyrddlgym-project/rddlrepository/tree/main/rddlrepository/archive/competitions/IPPC2023/ Strong foundation for RDDL implementation. Key differences:

- System flow is *node-centric* - the release() action cannot control proportion of released water sent to 2+ downstream reservoirs (instead simply proportional). Our model needs to control fluid movement between nodes directly by operating on connections (making it *arc-centric*).
- Stochastic addition/reduction of Reservoir level (by rain and evaporation respectively). Volumetric flow must be conserved throughout our model, so stochasticity will need to be introduced to the simulation elsewhere (specifically through flow resistance due to pressure).
- All levels are known by the agent (fully-observable). We need to modify the model to function off incomplete information provided by sensors, to provide a partial state representation the model can build from.

6. Summary

6.1. Future Work

- Extend the domain space to function with partially-observed domains.
- Run the planner on a more robust simulator that can specify other parameters (different tank elevations, turbulent flow, etc.)
- Perform hyperparameter tuning on important values such as reward coefficients, polling rate/training time, utility thresholds for optimal performance.
- Train a deep policy network on complex problem instances and compare performance vs linear replanning.
- Build an automated system that modifies existing targets and/or creates new targets during evaluation process to see how quickly planners can respond.