Building Water Management System

Authors

1. Renan Luiz Arceno
2. André Henrique Gomes
3. Thibault Fruchart

Motivation

We will continue the previous implementation of water management system, developed by students and can be accessed on this link Hydraulic System.
The main motivation of this project is to provide intelligent management for one of the main resources of any property, water. With this, it will be possible to detect leaks and take the necessary measures, thus reducing the financial, material and ecological damage caused by these incidents.
Another motivation and the ability to make the project sensors independently energetic. This will allow the sensors to be placed in a greater variety of places and will also reduce the costs of their installation, as there would be no need to disturb the building's electrical grid.

Goals

Our main goal in this project is to continue the work of the previous group.Hydraulic System.
They just made the theoric study on the subject, creating fake datas to test and generating pseudo algorithms. We will implement their solution focusing on the water leaking problem. Also we will try to generate electrical energy, taking advantage of the mechanical energy generated by the turbine in the water sensor, to power the sensor.

Methodology

To accomplish the goal of this project, we will follow those steps:

- **Studying**: reading about the problem, reading the older project to know how to implement the solutions. Also we need to read about the energy solution we propose, to know if it's really possible to do.
- **Coding**: coding the solution on an EPOS Mote connected with a water leak sensor, utilizing a neural network to distinguish an actual leak from a normal day of sewer waste.
- **Testing**: testing the code if its working as expected.

Tasks

1. Search the literature on how to implement water leakage sensors in EPOS Mote and how previous projects were done.
2. Search for possible ways of implementing the energy supply of sensors (energy harvesting).
3. Implementing our own server to receive the data through an EPOS that will serve as gateway and execute the neural network algorithm to check if its a leak or not.
4. Implementation the capture of the sensor data gathered by the water flow sensor connected on EPOS and send this data to our personal server.
5. Implementation of the project, code with the implementation of the data analysis of the sensors and partial implementation of the energy supply of the sensor through the water turbine.

Deliverables

1. A step-by-step report on how energy convertion works and how to implement it on our project.
2. Our own server that will the neural network algorithm working and the gateway receiving data from sensors.
3. Partial implementation of the energy harvesting mechanism and sensors sending data to our gateway.
4. Complete project code and complete report of everything done during development, including tests.

Schedule

Summarize tasks and deliverables on a weekly time table.

Task	18/09	25/09	02/10	09/10	17/10	24/10	02/11	09/11	23/11
Task1									
Task2	x	x	D1						
Task3			x	x	x	D2			
Task4					x	x	x	x	D3
Task5							x	x	xD4

Basic Concepts

Energy Harvesting

Energy harvesting is a method of powering wireless electronic device by harvesting ambient energy sources such as environmental vibrations, solar, radio frequency and human power. Harvesting energy from the environment is not new but it has been used from hundreds of years to generate energy from wind and water flow.
In order to be cost effective in many applications, the sensor nodes must be low cost and low maintenance. This presents challenges in terms of sensor calibration, packaging for survival in harsh environments and, particularly, the efficient supply and utilization of power. While the performance of battery technology is gradually improving and the power requirements of electronics is generally dropping, these are not keeping pace with the increasing demands of many WSN applications. For this reason, there has been considerable interest in the development of systems capable of extracting useful electrical energy from existing environmental sources. Another advantage is to allow the sensors to be placed in places of more difficult access, since there will be less need for maintenance and there will be no need to replace the batteries of the sensors when they are exhausted.
There are several types of energy that can be harvested:

- **1 Electromagnetic radiation**

The electromagnetic spectrum contains regions where ambient energy levels are very high and other regions where the ambient levels are typically much lower. The efficiency of conversion into electrical energy also depends on the part of the spectrum considered.
Photovoltaic conversion of visible light to electrical power is well established and photovoltaic devices provide relatively high efficiency over a broad range of wavelengths. These devices are typically low cost and provide voltage and current levels that are close to those required for microelectronic circuits.
Although radio frequency signals can be used to power passive electronic devices such as radio frequency identification (RFID) tags, these must be carefully tuned to the frequency of the radio source and are typically only capable of transmitting power over a distance of a few meters.
Without the use of such a dedicated source of radio frequency energy, the ambient levels are very low and are spread over a wide spectrum. Harvesting useful levels of electrical energy in these ambient conditions would require large broadband antennas.

- **2 Thermal**

Extraction of energy from a thermal source requires a thermal gradient. The efficiency of conversion from a thermal source is limited by the Carnot efficiency to **n <= (Th – Tc)/ Th**.Where Th is the absolute temperature on the “hot” side of the device, and Tc is the absolute temperature on the “cold” side. Thus,

the greater the temperature difference, the greater the efficiency of the energy conversion.

- **3 Mechanical energy sources**

Sources of mechanical energy may usefully be grouped as those dependent on motion which is essentially constant over extended periods of time, such as air flow used in a turbine, those dependent on intermittent motion, such as human footfall and those where the motion is cyclic, as in vibration sources.

- **3.1 Steady state mechanical source**

Sources of ambient energy which are essentially steady state are based around fluid flow, as in wind and air currents and water flow either in natural channels or through pipes, or around continuous motion of an object such as a rotating shaft

- **3.2 Intermittent mechanical sources**

Energy is available from motion which may be cyclic in nature but in which the energy is only available for a short part of the cycle. Examples of this type include energy available from vehicles passing over an energy harvesting device and intermittent human activity such as walking

- **3.3 Vibration**

Vibration energy is available in most built environments. The energy that can be extracted from a vibration source depends on the amplitude of the vibration and its frequency. It also depends on the extent to which the presence of an energy harvesting device affects the vibration. This, in turn, depends on the mass of the harvesting device relative to that of the vibrating mass. The energy that can be extracted from a vibration source depends on the frequency and amplitude and, since the majority of vibration based conversion devices have a relatively narrow range of operating frequencies, it is important that the nature of the source be understood.

Artificial Neural Network

Artificial neural networks (ANNs) are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn to do tasks by considering examples, generally without task-specific programming.
An ANN is based on a collection of connected units called artificial neurons, each connection or synapse between neurons can transmit a signal to another neuron. The receiving neuron can process the signal and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times. In artificial networks with multiple hidden layers, the initial layers might detect primitives and their output is fed forward to deeper layers who perform more abstract generalizations and so on.

Components os a Artificial Network

- **Neurons**

A neuron with label j receiving an input $\boldsymbol{p(t)}$ from predecessor neurons consists of the following components:
• an activation $\boldsymbol{a(t)}$, depending on a discrete time parameter;
• possibly a threshold ,$\boldsymbol{\theta}$ which stays fixed unless changed by a learning function;
• an activation function $\boldsymbol{f}$ that computes the new activation at a given time $\boldsymbol{t+1}$ from $\boldsymbol{a(t)}$, $\boldsymbol{\theta}$ and the input $\boldsymbol{p(t)}$ giving rise to the relation: $\boldsymbol{a(t+1) = f(a(t), p(t), \Theta)}$;

• and an output function ***fout*** computing the output from the activation.

- **Connections and weights**

The network consists of connections, each connection transferring the output of a neuron ***i*** to the input of a neuron ***j***. In this sense, ***i*** is the predecessor of ***j*** and ***j*** is the successor of ***i***. Each connection is assigned a weight ***w.***

- **Propagation function**

The propagation function computes the input ***p(t)*** to the neuron ***j*** from the outputs ***o(t)*** of predecessor neurons and typically has the form: ***p(t) =Σ o(t) w(i,j)***

- **Learning rule**

The learning rule is an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output This learning process typically amounts to modifying the weights and thresholds of the variables (i.e. the parameters of the ANN) within the network.

Neural networks learn things in exactly the same way, typically by a feedback process called **backpropagation**. This involves comparing the output a network produces with the output it was meant to produce, and using the difference between them to modify the weights of the connections between the units in the network, working from the output units through the hidden units to the input units.
The backpropagation algorithm can be described as follows:

1. There is normal propagation within the neural network and the initial results are obtained. In the first iteration the weights of the edges are usually initiated with random values;
2. The loss function is calculated from the difference between the obtained outputs and the expected outputs;
3. The propagation starts in the opposite direction, from the sida to the entrance of the neural network, using the expected outputs as the starting point. In this step the deltas of the paracada neuron outputs are calculated in the hidden layers of the neural network;
4. The deltas obtained in the previous step are multiplied with the activation inputs of each neuron to find the weight gradient;
5. A certain percentage of the weight gradient is added to the weight of the corresponding edge within the neural network. This percentage defines the speed at which the neural network will learn, the higher the percentage the faster the learning will be. However, slower learning often results in more accurate results;
6. Return to Step 1 using the new weights. The algorithm continues until the loss function reaches 0 or an acceptable margin.

Development

Sensors and Energy

After observing other examples of energy capture in hydraulics systems, we decided to opt for mechanical energy This type of energy source is the same used in hydroelectric plants, the water flow will rotate a turbine which in turn will move a dynamo that will generate electrical energy by electromagnetic induction. The difference is that this process will be done on a much smaller scale.

Model

The sensor chosen for this project will be the Yosoo DC Water Turbine Generator Water 12V DC 10W

Micro-hydro Water Charging Tool available from LISHA. It has been chosen because it has the ability to generate a quantity more than enough to supply the EPOS mote that will be used in the system.

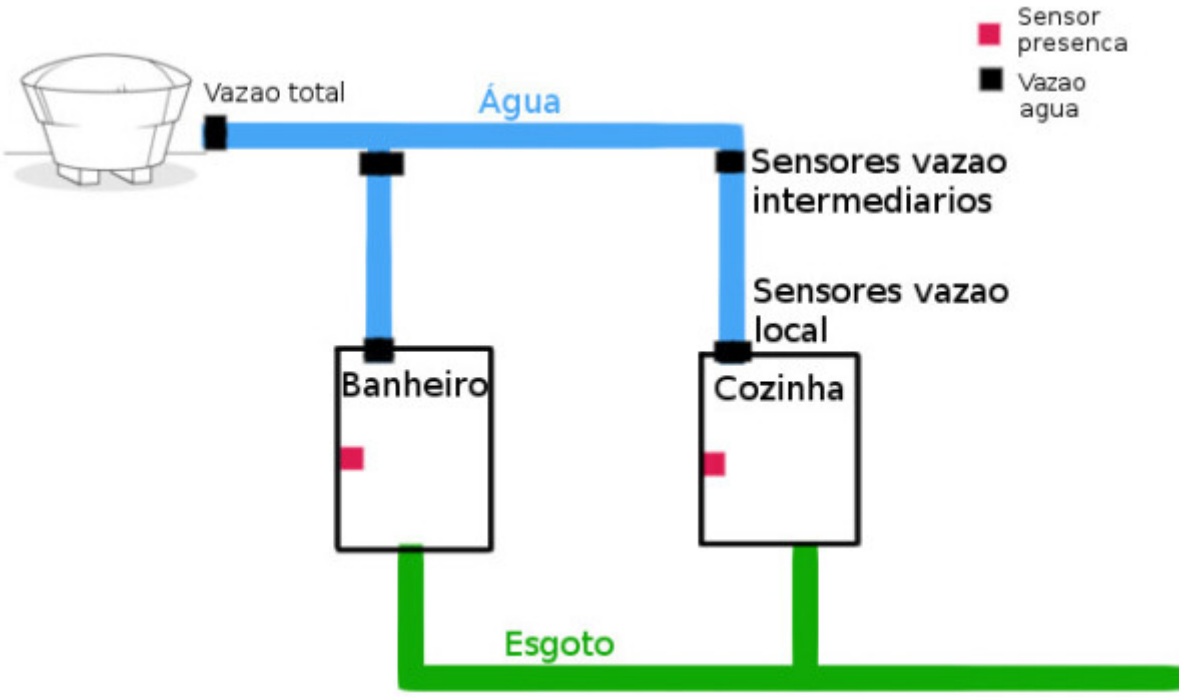
After sensor acquisition, we tested it in an uncontrolled environment to determine the voltage generated by the sensor under conditions of use. The test consisted primarily in creating a flow of water through the sensor, coupling two small pieces of pipe at the ends of the sensor to emulate the building's plumbing, and a voltmeter connected to the output designated to measure the voltage generated. Based on our tests we reached a minimum voltage of 0.2v and a maximum voltage of 12v.
The set also consists of an EPOS Mote connected to the hydraulic sensor to transmit ambient conditions to a gateway where a neural network will tell whether such a water flow characterizes a water leak or a use of the building's facilities.
However, because the sensor can generate more energy than the EPOS Mote can withstand (12v and 5v respectively) and the intermittent characteristics of the environment from which the energy will be captured, since the flow of water in certain parts is not continuous or has strength enough all the time to keep EPOS on all the time. There is a need for an additional piece of hardware to monitor the voltage coming out of the hydraulic sensor and ensure that there is enough energy to power EPOS, such hardware being the ICL7665 chip. As input this chip receives two values, one for the maximum voltage and the other for the expected minimum voltage which in this case are the results obtained in the tests of the hydraulic sensor.
Originally this part of energy management would be made by two transistors, one of 3v and another of 5v, and a scheme of check points to maintain the consistency of the readings of the sensor. However, after a conversation with the teacher, we chose to use the ICL7665 chip provided by LISHA, since the use of capacitors and check points would add unnecessary difficulties to the development of the project.

Another part of the system is the presence sensors present in the locations where there is water use in the building (bathrooms, kitchens, service areas, etc.). Its role is to try to associate a stream of water in one of these locations with the presence of one or more individuals, if there is anyone present and there is a flow of water in that location there is probably someone using the water and there is not a leak at the moment. These are common presence sensors present in the market, and can operate based on movement or temperature.
In a new conversation the professor pointed out that the use of a conventional presence sensor can hide leaks when used in high-traffic environments such as a shopping mall. The following situation exemplifies this:
Imagine that in one of several toilets in a mall there is a broken faucet and it is leaking water continuously. Bathroom users come in, look at the broken faucet and ignore the leak since it is not their

problem, so do the cleaners and other mall staff members.
In this example we have a leak occurring in conjunction with the presence of people, which in our system does not characterize a leak.
So how would we identify the leak in this situation?
To answer this question we have been suggested the use of UWB (ultra-wideband) sensors for the detection of people within water use points. This device can count people in a room with a good precision thanks to radio technology (this device uses severals signals of differents frequencies), when people reflect the signal waves that the UWB sensor emit, the UWB sensor recept it and the signal waves are analyzed by the algorithm, on the output of this algorithm we got the number of people in front of the sensor.

This type of sensor would help to solve the problem caused by the use of conventional presence sensors since we would have more accurately the number of people inside the resynthesis and we could verify if the water flow corresponds to the number of people present.
Unfortunately we can not acquire one of these sensors to use in system development, but they show great potential for possible future work related to this topic.

Model Operation



The principle of operation of the model is given by comparing the water flows between the sensor present at the water outlet of the building and the sensors at the points of use (bathrooms, kitchens, etc.). Based on this comparison we can reach the following senarios:

1. There is flow out of the water box, there is flow at points of use and there are people present at these points. Soon there is no leak;

1. There is flow out of the water box, there is flow at the points of use and there are no people present at these points. Soon there is a leak;

1. There is flow out of the water box and there is no flow at the points of use. Soon there is a leak;

1. There is no flow at the outlet of the water box, there is flow at the points of use and there are people present at these points. Soon there is no leak;

1. There is no flow at the outlet of the water box and there is no flow at the points of use. Senary not

deterministic, since there may be a malfunction in the pipes, but there is no water passing through them. By convention we classify as not being a leak.

Originally there was a second class of hydraulic sensors present at strategic points in the pipeline between the water box outlet and the points of use. These sensors would point more accurately at which part of the barrel the leak is occurring. But when talking to the teacher, we were pointed to a serious problem with this solution: this class of sensors would be inside the walls and when compared to the plumbing itself they do not have such a long shelf life, as a result there would be a need to break the walls of the address to a certain frequency simply to ensure the operation of these sensors. This would not only result in a high cost for maintaining the system as a major inconvenience to users.
Another factor unfavorable to this solution is that the plumbing of the building where the system will possibly be used is the sample, not inside the walls, which facilitates the location of leaks.
With this in mind we decided not to opt for the use of this class of sensors to locate leaks and we only have the sensors present at the exit of the water box and at the points of use of the building.

Computational Model

As we did not have a presence sensor and the flow sensor we had could not be directly connected to the EPOS because of the voltage it generates, we used EPOS itself as a gateway to generate signals to simulate a real sensor so the tests could be made. EPOS pins D0, D1, D2, D3, D4 and A7 were used.
Even without using real sensors, we still need to create Transducers for the "virtual sensors" we use via EPOS itself. A Transducer class interfaces to hardware mediator for a transducer (a sensor and / or actuator) with a SmartData instance.

Flow Sensor Transducer:

```
class WaterFlowSensor { public: typedef _UTIL::Observed Observed; typedef _UTIL::Observer Observer; static const unsigned int UNIT = TSTP::Unit::Electric_Current; static const unsigned int NUM = TSTP::Unit::I32; static const int ERROR = 0; static const bool INTERRUPT = false; static const bool POLLING = true; public: WaterFlowSensor() { } /* Since we couldn't test our solution on a real sensor, we are sending fake data. */ static void sense(unsigned int dev, Smart_Data<WaterFlowSensor> *data) { data->_value = Random::random() % 101; } static void actuate(unsigned int dev, Smart_Data<WaterFlowSensor> * data, const Smart_Data<WaterFlowSensor>::Value & command) {} static void attach(void *x){} static void detach(void *x){} };
```

Presence Sensor Transducer:

```
class PresenceSensor { public: typedef _UTIL::Observed Observed; typedef _UTIL::Observer Observer; static const unsigned int UNIT = CUSTOM_UNITS::UNIT_SWITCH; static const unsigned int NUM = TSTP::Unit::I32; static const int ERROR = 0; static const bool INTERRUPT = false; static const bool POLLING = true; public: PresenceSensor() { } /* Since we couldn't test our solution on a real sensor, we are sending fake data */ static void sense(unsigned int dev, Smart_Data<PresenceSensor> *data) { data->_value = Random::random() % 2; } static void actuate(unsigned int dev, Smart_Data<PresenceSensor> * data, const Smart_Data<PresenceSensor>::Value & command) {} static void attach(void *x){} static void detach(void *x){} };
```

Main:

```
int main() { WaterLeakFlow waterSensor(1, 100, WaterLeakFlow::Mode::ADVERTISED);
PeoplePresence presenceSensor(1, 100, PeoplePresence::Mode::ADVERTISED);
Thread::self()->suspend(); return 0; }
```

Neural Network

pessoas	maquinas	vazao_total	vazao_1	sensor_p1	vazao_2	sensor_p2	vazao_3	sensor_p3	vazamento	t_seg	hora
0	4	0.36	0.11	0	0.13	0	0.13	0	0	30	00:00:30
0	2	0.16	0.05	0	0.06	0	0.05	0	0	90	00:01:30
0	5	0.49	0.18	0	0.14	0	0.17	0	0	150	00:02:30
0	5	0.46	0.20	0	0.13	0	0.14	0	0	210	00:03:30
0	1	0.10	0.04	0	0.03	0	0.03	0	0	270	00:04:30
0	0	0.00	0.00	0	0.00	0	0.00	0	0	330	00:05:30
0	3	0.31	0.08	0	0.11	0	0.12	0	0	390	00:06:30
0	3	0.27	0.12	0	0.08	0	0.08	0	0	450	00:07:30
0	3	0.32	0.09	0	0.11	0	0.12	0	0	510	00:08:30
0	0	0.00	0.00	0	0.00	0	0.00	0	0	570	00:09:30
0	5	0.47	0.14	0	0.13	0	0.20	0	0	630	00:10:30
0	2	0.19	0.06	0	0.06	0	0.06	0	0	690	00:11:30
0	4	0.36	0.11	0	0.13	0	0.12	0	0	750	00:12:30
0	3	0.27	0.12	0	0.08	0	0.08	0	0	810	00:13:30

Before going into detail how the neural network was created, let's first take a look at the data set on which it will work.
The data set consists of:

1. People (Pessoas): represents the number of people registered and who are present in the building at that particular moment;
2. Machines(Máquinas): represents the number of machines that make use of water (washing machines, automatic irrigators, fountains, etc.) do not need the presence of a person to function;
3. Total flow (Vazão_total): represents the sensor present at the outlet of the water box;
4. Flow(vazão): represents the flow sensor at a certain point of water use in the building, in the example there are 3 sensors (vazão_1, vazão_2, vazão_3);
5. Sensorp: represents the presence sensors at a certain point of use of water in the building, they will always be used together with their respective flow sensor, for example: sensorp_1 will be associated with the flow sensor_1 and so on. When a sensor has a value of 0 it means that there are people at that point of use and when the value is 1 it means that there are no people at that moment at a certain point of use.
6. Leak(Vazamento): Contains the value that indicates whether the data set characterizes a leak or not;
7. Time (Tempo): is the interval between system status catches. In the example it has a set value of 30 seconds.

The data is arranged in table form inside the server which in turn will be used as input to the neural network. In this table, each column represents one of the previously described data while the lines represent states of the system over time, each row can be thought of as a photo of the system at a given time **t**.

For the implementation of the neural network we use the TensorFlow open source library that makes use of data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges

represent the multidimensional data arrays (tensors) communicated between them.

The central unit of data in TensorFlow is the tensor. A tensor consists of a set of primitive values shaped into an array of any number of dimensions. A tensor's rank is its number of dimensions. Here are some examples of tensors:

```
3 # a rank 0 tensor; a scalar with shape [] [1., 2., 3.] # a rank 1 tensor; a vector with shape [3] [[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3] [[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

TensorFlow Core programs as consisting of two discrete sections:

1. Building the computational graph.
2. Running the computational graph.

A computational graph is a series of TensorFlow operations arranged into a graph of nodes. Each node takes zero or more tensors as inputs and produces a tensor as an output. One type of node is a constant. Like all TensorFlow constants, it takes no inputs, and it outputs a value it stores internally.
A graph can be parameterized to accept external inputs, known as placeholders. A placeholder is a promise to provide a value later. To make the model trainable, we need to be able to modify the graph to get new outputs with the same input. Variables allow us to add trainable parameters to a graph, they are constructed with a type and initial value.
Here is the code for the neural network constructed using the TensorFlow library:

```
from __future__ import absolute_import from __future__ import division from __future__ import print_function import os import itertools import pandas as pd import tensorflow as tf # tf.logging.set_verbosity(tf.logging.INFO) COLUMNS = ["pessoas", "maquinas", "vazao_total", "vazao_1", "sensor_p1", "vazao_2", "sensor_p2", "vazao_3", "sensor_p3", "vazamento", "t_seg", "hora"] FEATURES_VAZ = ["pessoas", "maquinas", "vazao_total", "vazao_1", "sensor_p1", "vazao_2", "sensor_p2", "vazao_3", "sensor_p3", "vazamento"] FEATURES = ["pessoas", "maquinas", "vazao_total", "vazao_1", "sensor_p1", "vazao_2", "sensor_p2", "vazao_3", "sensor_p3"] LABEL = "vazamento" def get_input_fn(data_set, num_epochs=None, shuffle=True): return tf.estimator.inputs.pandas_input_fn( x=pd.DataFrame({k: data_set[k].values for k in FEATURES}), y=pd.Series(data_set[LABEL].values), num_epochs=num_epochs, shuffle=shuffle) def predict_value(pessoas, maquinas, vazao_total, vazao_1, sensor_p1, vazao_2, sensor_p2, vazao_3, sensor_p3, vazamento): # Load datasets training_set = pd.read_csv("hidro2_train.csv", skipinitialspace=True, skiprows=1, names=COLUMNS) test_set = pd.read_csv("hidro2_test.csv", skipinitialspace=True, skiprows=1, names=COLUMNS) dados = dict(zip(FEATURES_VAZ, [pessoas, maquinas, vazao_total, vazao_1, sensor_p1, vazao_2, sensor_p2, vazao_3, sensor_p3, vazamento])) predict_func = tf.estimator.inputs.pandas_input_fn( x=pd.DataFrame([dados], columns=dados.keys()), y=pd.Series(dados[LABEL]), num_epochs=1, shuffle=False) # Feature cols feature_cols = [tf.feature_column.numeric_column(k) for k in FEATURES] # Build 2 layer fully connected DNN with 30, 30 units respectively. model_folder = "/tmp/hidro" regressor = tf.estimator.DNNRegressor(feature_columns=feature_cols, hidden_units=[25, 30], model_dir=model_folder) # Train if we dont have a trained model if not os.path.isdir(model_folder) and not os.path.exists(model_folder): regressor.train(input_fn=get_input_fn(training_set), steps=5000) # Evaluate loss over one epoch of test_set. ev = regressor.evaluate(input_fn=get_input_fn(test_set, num_epochs=1)) loss_score = ev["loss"] print("Loss: {0:f}".format(loss_score)) # Print out predictions y = regressor.predict(input_fn=predict_func) # .predict() returns an iterator of dicts; convert to a list and print predictions = list(p["predictions"] for p in itertools.islice(y, 1)) print("Predictions: {}".format(str(predictions[0][0]))) return loss_score, predictions[0][0]
```

In our model data input to the neural network is in the form of a table and to use it as an efficient table we have to create columns for this data.

```
COLUMNS = ["pessoas", "maquinas", "vazao_total", "vazao_1", "sensor_p1", "vazao_2", "sensor_p2", "vazao_3", "sensor_p3", "vazamento", "t_seg", "hora"] LABEL = "vazamento"
```

Because the task is a binary sort, we construct a label to identify leaks. The label "leak" contains the value of 0 when there is no leak and 1 when there is a leak.

The next step is to convert the data to a form suitable for processing in the neural network.
When building a tf.estimator model, the input data is specified by means of an Input Builder function. This builder function will not be called until later passed to tf.estimator.Estimator methods such as train and evaluate. The purpose of this function is to construct the input data, which is represented in the form of Tensors

```
def get_input_fn(data_set, num_epochs=None, shuffle=True): return tf.estimator.inputs.pandas_input_fn( x=pd.DataFrame({k: data_set[k].values for k in FEATURES}), y=pd.Series(data_set[LABEL].values), num_epochs=num_epochs, shuffle=shuffle)
```

Next we need to define how the neural network will work, for this we use the DNNRegressor. The DNNRegressor is API to jointly train a wide linear model and a deep feed-forward neural network. This approach combines the strengths of memorization and generalization.
But first we need to transform raw data into appropriate input resources, this is done through FeatureColumn. A FeatureColumn represents a single feature in your data.

```
FEATURES = ["pessoas", "maquinas", "vazao_total", "vazao_1", "sensor_p1", "vazao_2", "sensor_p2","vazao_3", "sensor_p3"] feature_cols = [tf.feature_column.numeric_column(k) for k in FEATURES]
```

Once this is done, we define the regression model using the previously created FeatureColumns. In our model we opted for a neural network of two fully connected layers containing 30 neurons each (this value was reached by testing different values and checking the loss output), the model will be saved in the folder / tmp / hidro /, this was done to avoid having to train the network every time the algorithm was executed.

```
regressor = tf.estimator.DNNRegressor(feature_columns=feature_cols, hidden_units=[30, 30], model_dir="/tmp/hidro")
```

Finally, there is only training and evaluation of the network, this is done using a training set and a test set.

```
training_set = pd.read_csv("hidro2_train.csv", skipinitialspace=True, skiprows=1, names=COLUMNS) test_set = pd.read_csv("hidro2_test.csv", skipinitialspace=True, skiprows=1, names=COLUMNS) regressor.train(input_fn=get_input_fn(training_set), steps=1000) ev = regressor.evaluate(input_fn=get_input_fn(test_set, num_epochs=1))
```

The output of the network evaluation presents a loss that is nothing more than the inactivity of the network.
To train and test our DNN we created a simulation which can be found in this link
The final code can be found in our Github page

Server

For the tests we created a simple server with only a GET method that serves only to send the parameters to the neural network, as a result ofisot is returned the label containing 0 if there is no leak and 1 case there is leakage and the value of the network inactivity.
For communication, the http request framework called Tornado was used.

```
```

```
import tornado.ioloop import tornado.web import hidro_neural as neural import csv import time import json import sqlite3 as db def get_int_default_from_file(): original_filename = "hidro2.csv" with open(original_filename, "r") as file: reader = csv.DictReader(file) now = int(time.time()) now = now % (24 * 60 * 60) for row in reader: row_seg = int(row['t_seg']) if (now - 20) <= row_seg <= (now + 20): return int(row['pessoas']), int(row['maquinas']) def check_vazamento(maquinas, pessoas, sensor_p1, sensor_p2, sensor_p3, vazamento, vazao_1, vazao_2, vazao_3, vazao_total): loss, prediction = neural.predict_value(pessoas, maquinas, vazao_total, vazao_1, sensor_p1, vazao_2, sensor_p2, vazao_3, sensor_p3, vazamento) vazando = abs(round(prediction)) return loss, vazando class MainHandler(tornado.web.RequestHandler): def get(self): """ http://localhost:8080/ prediction?pessoas=53 &maquinas=2 &vazao_total=2.27 &vazao_1=0.78 &sensor_p1=0 &vazao_2=0.78 &sensor_p2=0 &vazao_3=0.78 &sensor_p3=0 &vazamento=0 """ pessoas = int(self.get_query_argument("pessoas", default=-1)) maquinas = int(self.get_query_argument("maquinas", default=-1)) vazao_total = float(self.get_query_argument("vazao_total", default=0)) vazao_1 = float(self.get_query_argument("vazao_1", default=0)) sensor_p1 = int(self.get_query_argument("sensor_p1", default=0)) vazao_2 = float(self.get_query_argument("vazao_2", default=0)) sensor_p2 = int(self.get_query_argument("sensor_p2", default=0)) vazao_3 = float(self.get_query_argument("vazao_3", default=0)) sensor_p3 = int(self.get_query_argument("sensor_p3", default=0)) vazamento = int(self.get_query_argument("vazamento", default=0)) if pessoas == -1 or maquinas == -1: pessoas_aux, maquinas_aux = get_int_default_from_file() if pessoas == -1: pessoas = pessoas_aux if maquinas == -1: maquinas = maquinas_aux loss, vazando = check_vazamento(maquinas, pessoas, sensor_p1, sensor_p2, sensor_p3, vazamento, vazao_1, vazao_2, vazao_3, vazao_total) if vazando == 0: self.write("<font color=\"blue\">NÃO</font> está com vazamento") else: self.write("<font color=\"red\">ESTÁ</font> com vazamento") self.write("<br>") self.write("Entropia rede neural: " + str(loss)) class PostHandler(tornado.web.RequestHandler): def post(self): data = json.loads(self.request.body.decode('utf-8')) con = db.connect('water_flow') con.executemany("INSERT INTO sensor_data(data, x, y, z, valor, unidade) VALUES (?,?,?,?,?,?)", (data['data'], data['x'], data['y'], data['z'], data['valor'], data['unidade'])) con.commit() con.close() class CheckHandler(tornado.web.RequestHandler): def get(self): con = db.connect('water_flow.db.db') cursor = con.cursor() cursor.execute("SELECT * FROM sensor_data WHERE checado = 0 ORDER BY _id DESC LIMIT 7") # in our example, theres 7 sensors sensor_values = [] for entry in cursor.fetchall(): sensor_values.append(entry[5]) # 5 = valor pessoas_aux, maquinas_aux = get_int_default_from_file() loss, vazando = check_vazamento(maquinas_aux, pessoas_aux, sensor_values[0], sensor_values[1], 0, sensor_values[2], sensor_values[3], sensor_values[4], sensor_values[5], sensor_values[6]) con.close() if vazando == 0: self.write("<font color=\"blue\">NÃO</font> está com vazamento") else: self.write("<font color=\"red\">ESTÁ</font> com vazamento") self.write("<br>") self.write("Entropia rede neural: " + str(loss)) def make_app(): return tornado.web.Application([ (r"/prediction", MainHandler), (r"/put", PostHandler), (r"/check", CheckHandler), ]) if __name__ == "__main__": app = make_app() app.listen(8080) print("Server started") tornado.ioloop.IOLoop.current().start()
```

Gateway

```
```

```
#include <smart_data.h> #include <utility/ostream.h> #include <alarm.h> #include <thread.h> using namespace EPOS; IF<Traits<USB>::enabled, USB, UART>::Result io; const TSTP::Time DATA_PERIOD = 1 * 1000000; const TSTP::Time DATA_EXPIRY = 3 * DATA_PERIOD; const TSTP::Time INTEREST_EXPIRY = 2ull * 60 * 60 * 1000000; int main() { Alarm::delay(DATA_PERIOD*10); // Get epoch time from serial TSTP::Time epoch = 0; char c = io.get(); if(c != 'X') { epoch += c - '0'; c = io.get(); while(c != 'X') { epoch *= 10; epoch += c - '0'; c = io.get(); } TSTP::epoch(epoch); } // Interest center points TSTP::Coordinates center(0, 0, 0); // Regions of interest TSTP::Time start = TSTP::now(); TSTP::Time end = start + INTEREST_EXPIRY; TSTP::Region region(center, 0, 600, -1); // Data of interest PeoplePresence presenceSensor(region, DATA_EXPIRY, DATA_PERIOD); WaterLeakFlow waterSensor(region, DATA_EXPIRY, DATA_PERIOD); // Time-triggered actuators while(TSTP::now() < end) { Alarm::delay(DATA_PERIOD); PeoplePresence::DB_Record w_record = waterSensor.db_record(); WaterLeakFlow::DB_Record p_record = presenceSensor.db_record(); cout << "{'value':'" << (float) w_record.value << "',"; cout << "'x':'" << w_record.x << "',"; cout << "'y':'" << w_record.y << "',"; cout << "'z':'" << w_record.z << "'}" << endl; cout << "{'value':'" << (float) p_record.value << "',"; cout << "'x':'" << p_record.x << "',"; cout << "'y':'" << p_record.y << "',"; cout << "'z':'" << p_record.z << "'}" << endl; } return 0; }
```

Improvements

Some improvements can be made to the model to circumvent some problems or improve their performance. Here are some improvements that can be made:

Use of a UWB sensor in place of common presence sensor

As previously shown in the section Sensor and Energy, this substitution would serve to have a better control of how many people are present in the environment at a given moment and to better distinguish between situations of water leaks or specific use of water resources. In UWB human detection radar system: the RF CMOS chip and algorithm integrated sensor {13} we find an example of the use of this type of sensor for identification and counting of people in certain environment.
Obviously the addition of this type of sensor would also entail changing the neural network, since one of the inputs would be the number of people present in the place and not only if there are or not people in that place.

Addition of flow sensors in sewer pipes

These sensors, along with the others, would allow us to have more precise control of the amount of water used in the building, since we would know how much water came in and left the building. Obviously the number of people would need to be taken into account, since they tend to consume and leave water while they are using the building.

A Better simulation using data from a real building

Our simulation was made with fake datas. to improve our simulation we could monitor the sensors in a real building to better results.
At the moment our neural network uses only one neuron in the output layer, as a result the results are limited to only situations of leakage or non-leakage. While this is sufficient for small buildings, in larger and more moving environments there may be a need to rate the severity of a leak. We can obtain this using two neurons in the output layer of the neural network, one containing the weight of the probability of the situation being a leak and the other containing the probability of the situation not being a leak, comparing the weights of both neurons we obtain a larger spectrum of responses and we can also classify the severity of a leak when it runs. All this would ensure greater control over the building's water resources.

Bibliography

1. **Power Harvesting for Smart Sensor Networks in Monitoring Water Distribution System** - M. I. Mohamed, W. Y. Wu, and M. Moniri - 2011 International Conference on Networking, Sensing and Control Delft, the Netherlands : Year: April 2011 : Pages: 11-13
2. **Energy harvesting: State-of-the-art** - Adnan Harb - Renewable Energy 36 : Year: 2011 : Pages: 2641 - 2654.
3. **Comparison of Energy Harvesting Systems for Wireless Sensor Networks** - James M. Gilbert and Farooq Balouchi - International Journal of Automation and Computing, Volume 5, Issue 4 : Year: October :Pages: 334–347
4. **Maximize energy utilization for ultra-low energy harvesting powered embedded systems** - Chen Pan; Mimi Xie; Jingtong Hu - 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA):Year: 2017 :Pages: 1 - 6
5. **Challenges for Energy Harvesting Systems Under Intermittent Excitation** - Guang Yang, Bernard H. Stark, Simon J. Hollis, and Stephen G. Burrow - IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS, VOL. 4, NO. 3: Year: 2014.
6. **Power Management in the EPOS System** - Geovani Ricardo Wiedenhoft, Lucas Francisco Wanner, Giovani Gracioli and Antônio Augusto Fröhlich - In: SIGOPS Operating Systems Review, 42(6):71-80, 2008.
7. **TensorFlow** https://www.tensorflow.org/tutorials/
8. **TensorFlow Deep Learning** https://www.tensorflow.org/tutorials/wide_and_deep
9. **Neural Network** https://en.wikipedia.org/wiki/Artificial_neural_network
10. **Tornado** http://www.tornadoweb.org/en/stable/
11. **Python** https://www.python.org/about/gettingstarted/
12. **Pandas** https://pandas.pydata.org/
13. **UWB human detection radar system: A RF CMOS chip and algorithm integrated sensor** - SangHyun Chang; Ta-Shun Chu; Jonathan Roderick; Chenliang Du; Timothy Mercer; Joel. W. Burdick; Hossein Hashemi - 2011 IEEE International Conference on Ultra-Wideband (ICUWB): Year: 2011 : Pages: 355 - 359.