The Future of Engineering Education
2024 Annual Conference & Exposition
Oregon Convention Center
Portland, OR . June 23 - 26, 2024
ASEE
Paper ID #42376

Work in Progress: Hardware-in-the-Loop Process Control Simulation Labs

Mr. Bradley Lane Kicklighter P.E., University of Southern Indiana

Brad holds a BS in Electrical Engineering from Rose-Hulman Institute of Technology (1989) and an MS in Electrical and Computer Engineering from Purdue University (2001).

His past work experience includes eleven years at Delphi (formerly Delco Electronics) as an Advanced Project Engineer, eleven years at Whirlpool Corporation as a Lead Engineer/Solution Architect, and three years at Ivy Tech Community College as an Instructor/Program Chair Pre-Engineering. Since 2015, he has been employed at the University of Southern Indiana as a Clinical Associate Professor of Engineering Technology.

He holds three patents, has served as an IEEE section officer since 2004, and has been a Licensed Professional Engineer in the State of Indiana since 2005.

Work in Progress: Hardware-in-the-Loop Process Control Simulation Labs

Abstract

An automation course typically covers the topic of process control. If available, this involves the use of a physical process trainer for each individual student. Our university is limited to just four trainers for an entire class. This leads to students being teamed up on a process trainer which can limit the hands-on experience for some students. To achieve a positive experience for all students, while covering the same objectives as having a physical trainer for each student, a hardware-in-the-loop (HIL) process control simulation has been implemented. This allows for 1. Individualized learning, 2. An ability to cover the objectives without having physical trainers, and 3. Instructs students about hardware-in-the-loop simulation.

This paper presents the work to date on hardware-in-the-loop (HIL) process control simulation labs for SCADA (Supervisory Control and Data Acquisition) Systems Design. The implementation uses a programmable logic controller (PLC) for the controller and a simulation of the process (plant) written in Python by the author running on a desktop computer. The desktop computer and the PLC communicate via Modbus. The subject of the simulation is control of water level in a tank.

The course is open to all juniors and seniors in all programs in the Engineering Department. It covers programming of PLCs, process control, and industrial networks.

Direct assessment in the form of labs graded with a rubric and indirect assessment in the form of an end of semester survey of learning outcomes is presented to illustrate student performance.

Introduction

Process control is a common topic in automation courses and the labs related to process control usually involve some sort of process trainer. Our department has four process trainers that allow control of water level, flow rate, pressure, and temperature. These trainers require Windows computers with control software provided by the trainer supplier to control the trainers. Since the automation course, SCADA Systems Design, can have up to sixteen students enrolled in it, there can be up to four students sharing a trainer.

The author has observed that with four students sharing a trainer, that sometimes not all students in a group have a good learning experience because one student may dominate the trainer or a student is willing to let others do all the work. Group work vs individual work is a strongly debated topic in higher education [1].

Another issue is that when our lab computers transitioned to Windows 10, the compatible version of the control software had a number of key features that no longer functioned, such as the ability to export data from a run.

These issues led the author to search for a different solution which resulted in the author writing a Python program to simulate water level in a tank. The simulation program runs on a personal computer (PC) and communicates with the programmable logic controller (PLC) in a PLC trainer. The communication link is Modbus RTU which is a protocol running on top of an EIA-485 (RS-485) serial link [2][3]. See Figure 1 for the system block diagram.

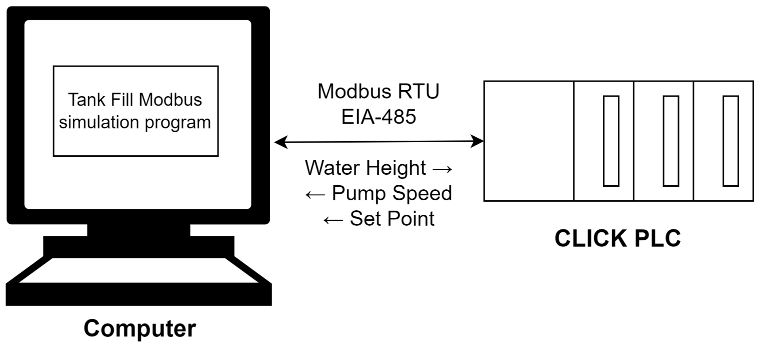


Figure 1: System block diagram of HIL simulation. The simulation program runs on the computer and communicates via the Modbus RTU protocol over an EIA-485 serial link to the PLC. A USB to RS-485 adapter is used to connect the computer to the PLC. The data exchanged between the simulation program and the PLC include water height, pump speed, and set point.

This arrangement of a PLC controlling a simulation of a process is an example of a hardware-in-the-loop (HIL) simulation where in this case the PLC is the hardware. HIL is used quite often in automotive testing [4]. A similar example from the literature of a PLC connected to an HIL simulation is one where the PLC is connected via OPC (OLE for Process Control) to FlexSim (3D simulation modeling and analysis software) [5].

Course

SCADA (Supervisory Control and Data Acquisition) Systems Design is an upper-division course taught to students in the Engineering Department as an elective or required course, depending upon the program. The following are the course learning outcomes:

1. Understand common Industrial Automation concepts, methods, and control algorithms.
2. Understand sensors and actuators used in Industrial Automation tasks.
3. Design Piping & Instrumentation Diagrams (P&IDs) for simple process systems.
4. Measure process variables in response to process parameters and analyze the resulting process behavior.
5. Understand Programmable Logic Controller (PLC) components, signal interface methods, and applications.
6. Design and write PLC control programs. Recognize other control program language formats defined in the IEC 61131-3 standard.
7. Design and program suitable Human Machine Interface systems.

8. Understand common industrial networking topologies, protocols, and hardware.

Methods

Since our engineering programs are ABET accredited, we collect data to support continuous improvement. Part of that data collection for this course includes a paper-based survey about the labs administered to the students at the end of the semester. It asks Likert scale questions about their knowledge about various topics learned in the labs along with free response questions about the labs and lab equipment. For this paper, the relevant survey questions include three Likert scale questions and four free response questions:

- I know how to perform on/off control.
- I know how to perform PID control.
- The Tank Fill Modbus program helped me learn about on/off and PID control.
- What did you like about the labs?
- What could be improved in the labs?
- What did you like about the lab equipment?
- What could be improved in the lab equipment?

The values for the Likert scale question responses are:

- 1 — Strongly Disagree
- 2 — Disagree
- 3 — Neither Agree nor Disagree
- 4 — Agree
- 5 — Strongly Agree

There are two labs that use the simulation program that is the subject of this paper:

- Lab 11: On/Off Control
- Lab 12: PID Control

Tables 1 and 2 show the rubrics used to score the two labs.

Table 1: Rubric for Lab 11: On/Off Control. The lab is worth a total of fifteen points.

Criteria	**Proficient**	**Competent**	**Novice**	**No Attempt**
Files Submitted	1	0.67	0.33	0
Program Demonstrated	1	0.67	0.33	0
Design Table	1	0.67	0.33	0
HMI Sketches	1	0.67	0.33	0
HMI Screens	1	0.67	0.33	0
HMI Screens	1	0.67	0.33	0
Calculations	1	0.67	0.33	0
Plot	1	0.67	0.33	0
Duty Cycle	1	0.67	0.33	0
Program Comments	1	0.67	0.33	0
Nicknames	1	0.67	0.33	0
Rung Comments	1	0.67	0.33	0
Initialization	1	0.67	0.33	0
Math	1	0.67	0.33	0
Start Stop	1	0.67	0.33	0
Pump States	1	0.67	0.33	0

Table 2: Rubric for Lab 12: PID Control. The lab is worth a total of fifteen points.

Criteria	**Proficient**	**Competent**	**Novice**	**No Attempt**
Files Submitted	1	0.67	0.33	0
Program Demonstrated	1	0.67	0.33	0
Design Table	1	0.67	0.33	0
HMI Sketches	1	0.67	0.33	0
HMI Screens	1	0.67	0.33	0
System Characterization Table	0.5	0.33	0.17	0
System Characterization Plot	0.5	0.33	0.17	0
Questions	0.5	0.33	0.17	0
Tuning Parameters	1	0.67	0.33	0
System Response Plot 1	0.5	0.33	0.17	0
System Response Plot 2	0.5	0.33	0.17	0
System Response Plot 3	0.5	0.33	0.17	0
System Response Plot 4	0.5	0.33	0.17	0
Questions	0.5	0.33	0.17	0
Program Comments	1	0.67	0.33	0
Nicknames	1	0.67	0.33	0
Rung Comments	1	0.67	0.33	0
Initialization	1	0.67	0.33	0
Math	1	0.67	0.33	0

The survey and lab score data are used to assess the effectiveness of the simulation program for learning about on/off control and Proportional-Integral-Deriviative (PID) control in this course.

The success criterion for the Likert scale questions is an average score that is above 3.0 (Neither Agree nor Disagree). The success criterion for the lab scores is for 70% or more students to score 70% or better.

Simulation Program

The simulation program was written by the author in Python and uses the Pymodbus library for Modbus RTU communication between the PC running the simulation program and the PLC [6][2][3]. The PC acts as a Modbus client and the PLC is a Modbus server. This means that the simulation program initiates all Modbus communication.

The simulation models the level of water in a tank where a variable-speed pump flows water into the top of the tank and water flows out of the bottom of the tank by gravity (see Figure 2).

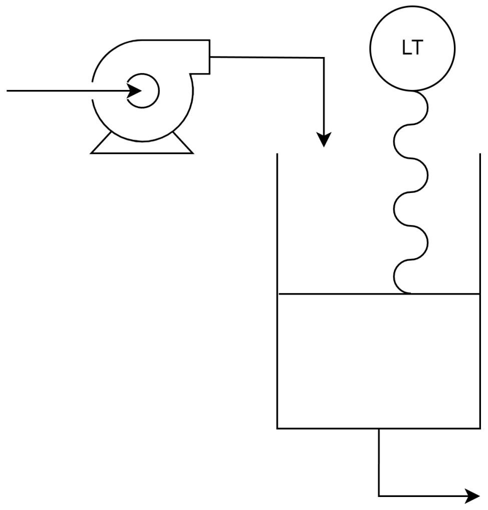


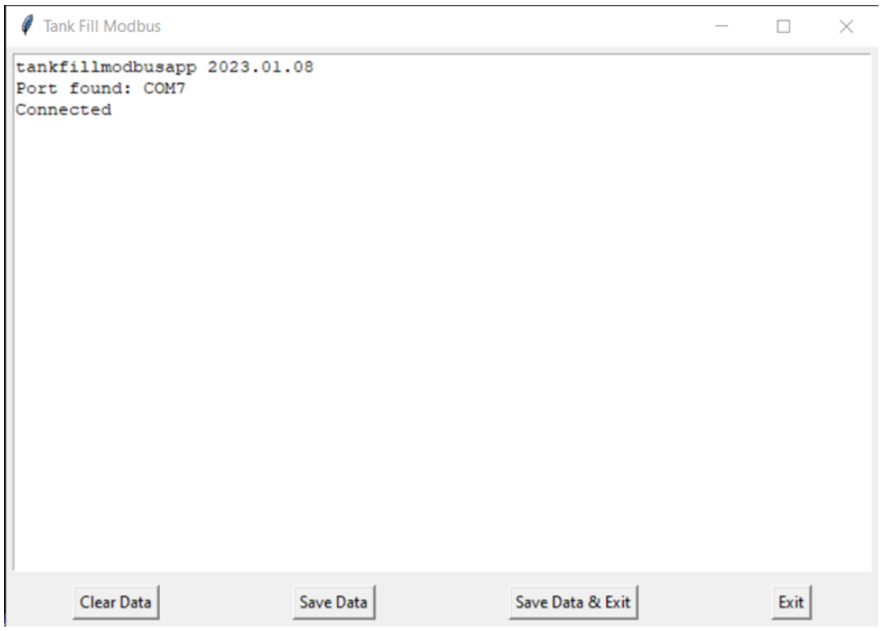
Figure 2: Process flow diagram showing the pump flowing water into an open tank with water flowing out of the bottom of the tank and an ultrasonic distance sensor measuring the water level in the tank.

Three variables are exchanged between the simulation program and the PLC via Modbus:

- Water Height — sent to PLC
- Pump Speed — sent to simulation program
- Set Point — sent to simulation program

The Set Point is sent to the simulation program so that it can be stored along with the other data for analysis afterward.

The simulation program has a simple graphical user interface with a message area and four buttons (see Figure 3).



```
tankfillmodbusapp 2023.01.08
Port found: COM7
Connected
```

Figure 3: The Tank Fill Modbus graphical user interface (GUI). Messages are displayed in the message area. The Clear Data button clears the collected data. The Save Data button saves the collected data to a comma-separated values (CSV) file. The Save Data & Exit saves the data and exits the program. The Exit button exits the program without saving data.

The comma-separated values (CSV) data file contains records of data consisting of time, pump speed, water height, and set point. In the labs, the students plot the data and use it to analyze the system.

See the appendix for the Python source code of the application and class that implements the simulation.

On/Off Control Lab

The objectives of the On/Off Control lab are:

- Create a process control system running in the on/off mode of control.
- Determine the deadband (differential gap) setting for an on/off control system.
- Determine the duty cycle of an on/off control system.
- Configure a PLC for Modbus RTU communication.

The students must write a ladder logic program for the PLC that implements on/off control of the water level in the simulated tank and create human machine interface (HMI) screens to run the system and plot data in real time. They must also configure the PLC for Modbus RTU communication on an EIA-485 serial link to the PC.

Given a desired differential gap of 8%, the students must calculate the deadband, enter this value on the HMI, run the system for at least ten cycles, save the data, create a plot of the data, and calculate the average duty cycle of the pump.

Proportional-Integral-Derivative (PID) Control Lab

The objectives of the PID Control lab are:

- Create a process control system running in the PID mode of control.
- Determine the PV vs CO relationship for a control system.
- Tune a PID control system using autotune.
- Observe the response of a PID control system to set point changes.
- Configure a PLC for Modbus RTU communication.

The students must configure the PLC to perform PID control of the water level in the simulated tank and create human machine interface (HMI) screens to run the system and plot data in real time. They must also configure the PLC for Modbus RTU communication on an EIA-485 serial link to the PC.

A CLICK PLC is used for the course and it has a PID control loop [7]. Before configuration of PID control is complete, the students must characterize the relationship between the control output (CO) and the process variable (PV) to ensure that it is linear and to determine the maximum expected PV value.

Once PID configuration is complete, the students run the autotune function to tune their systems. Lastly, the students must perform a series of set point step changes, capture the data, plot the response curves, and note their observations of the system behavior.

Results and Discussion

The simulation program was used for the on/off and PID control labs during two semesters: spring 2023 and fall 2023. Table 3 shows the number of students enrolled in each semester. All students during each semester completed the lab survey and the labs.

Table 3: Number of students enrolled in each semester. All students in each semester completed the labs survey and labs.

Semester	**Number of Students**
Spring 2023	11
Fall 2023	15
Total	**26**

The data from both semesters was combined to produce the following results. n = 26 for all results with the exception of "I know how to perform on/off control." where n = 25.

Likert Scale Survey Question Results

Table 4 shows the average scores for the Likert scale questions on the labs survey.

Table 4: Average scores out of 5 for the Likert scale questions from the labs survey.

Question	Average Score
I know how to perform on/off control.	4.27
I know how to perform PID control.	3.80
The Tank Fill Modbus program helped me learn about on/off and PID control.	4.23

Scores for all three questions ranged from 3 to 5.

Lab Score Results

Table 5 shows the results for the lab scores.

Table 5: Percentage of lab scores that are 70% or higher for the two labs.

Lab	Percentage
Lab 11: On/Off Control	100%
Lab 12: PID Control	92.3%

Free Response Survey Question Results

Of the free response questions, there were no responses relevant to the simulation program based labs for questions "What did you like about the labs?" or "What did you like about the lab equipment?".

For free response question "What could be improved in the labs?" there were two responses:

"I wish the tank fill modbus was more interactive."

"The tank program needs updated and struggles to work a lot of the time."

For free response question "What could be improved in the lab equipment?" there was one response:

"Add a physical unit to do on/off and PID control."

Conclusions and Recommendations

The success criterion for the Likert scale questions is an average score that is above 3.0. All three average Likert scale question scores were above 3.0 (the lowest was 3.80).

The success criterion for the lab scores is for 70% or more students to score 70% or higher. This criterion is met for both labs.

Based on the success criteria, it appears that HIL simulation may have been effective for student learning of on/off and PID control.

There are improvements to be made, however.

- As one student pointed out, the program needs to be more interactive. An animation of the water tank would be helpful as well as a real-time plot of the data. The real-time plots on the HMIs can only update once a second.
- One student had communication issues, which were due to a bad USB to RS-485 adapter. Higher quality cables will be investigated. Although, some systems showed many "re-connection" events, which could be due to either the Pymodbus library or the USB to RS-485 adapters. An alternate Modbus library will be investigated.
- If a small footprint, reasonably priced process trainer can be found, that would be preferable to the HIL simulation.

Despite its shortcomings, the HIL simulation is fulfilling the needs for these two process control labs.

References

[1] J. Belanger, "Learning in the Laboratory: Does Group Work Work?" Master's thesis, United States Military Academy, West Point, NY, 2015. [Online]. Available: https://www.westpoint.edu/sites/default/files/inline-images/centers_research/center_for_teching_excellence/PDFs/mtp_project_papers/Belanger_15.pdf

[2] "Modbus Application Protocol Specification," Andover, MA, Apr. 2012. [Online]. Available: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[3] "Modbus Over Serial Line Specification and Implementation Guide," Andover, MA, Dec. 2006. [Online]. Available: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

[4] "What is hardware-in-the-loop testing?" Mar. 2022. [Online]. Available: https://www.aptiv.com/en/insights/article/what-is-hardware-in-the-loop-testing

[5] K. Tebani, B. Mihoubi, M. Kadri, G. Mehdi, and B. Bouzouia, "Hardware-In-the-Loop Simulation for Validating PLC Programs," in *CIAM'15*, Oran Algérie, Nov. 2015.

[6] "Pymodbus-dev/pymodbus," pymodbus, Feb. 2024. [Online]. Available: https://github.com/pymodbus-dev/pymodbus

[7] "CLICK PLC: 4-pt in, 4-pt out, 4-ch in, 2-ch out, Ethernet and serial ports, (programmable logic controller) (PN# C0-12DD2E-2-D) | AutomationDirect." [Online]. Available: https://www.automationdirect.com/adc/shopping/catalog/programmable_controllers/click_plcs_(stackable_micro_brick)/plc_units/c0-12dd2e-2-d

Appendix

The following is the Python source code of the TankFillModbus application.

```
'''
TankFillModbus - Tank fill simulation that communicates with a PLC via Modbus RTU
tankfillmodbusapp.py

Brad L. Kicklighter, P.E.
1-3-2022
'''

__author__ = "Brad L. Kicklighter, P.E."
__version__ = "2023.01.08"

from tankfill import *
from pymodbus.client import ModbusSerialClient
from getRS485adapter import get_port
import time
import tkinter as tk
from tkinter import ttk

# Modbus RTU Constants
UNIT = 2  # Modbus server (PLC) node number
BAUDRATE = 38400  # Baud rate for port
BYTESIZE = 8  # Number of bits per character
PARITY = 'E'  # Parity for port
STOPBITS = 1  # Number of stop bits
TIMEOUT = 10  # Timeout in seconds
RETRYONEMPTY = True  # Retry transaction if empty response received
RECONNECTDELAY = 0  # Reconnect delay
# Tank Fill Simulation Constants
WATER_HEIGHT_ADDR = 0x0000
PUMP_SPEED_ADDR = 0x0001
SET_POINT_ADDR = 0x0002
TANK_DIAMETER = 0.25  # Diameter of tank (m)
TANK_HEIGHT = 0.3  # Height of tank (m)
TAU = 25  # Time constant of out flow of tank (s)
MAX_IN_FLOW = 0.5  # Maximum input flow rate to tank (kg/s)
TIME_STEP = 0.1  # Simulation time step (s)
PLC_MAX_PUMP_SPEED = 10000  # Maximum pump speed (PLC)
PLC_MAX_WATER_HEIGHT = 10000  # Maximum water height (PLC)

class App(tk.Tk):
def __init__(self):
```

```
tk.Tk.__init__(self)

# Attributes
self.adapter = None
self.client = None
self.t = None

# App Window
self.title('Tank Fill Modbus')
self.exiting = False
self.log = tk.Text(self, state='disabled', width=80, height=24, wrap='char')
self.log.grid(row=0, column=0, columnspan=4, padx=5, pady=5)
self.clear_btn = tk.Button(self, text='Clear Data', command=self.on_clear_data)
self.clear_btn.grid(row=1, column=0, padx=5, pady=5)
self.save_btn = tk.Button(self, text='Save Data', command=self.on_save_data)
self.save_btn.grid(row=1, column=1, padx=5, pady=5)
self.save_exit_btn = tk.Button(self, text='Save Data & Exit',
  command=self.save_exit_app)
self.save_exit_btn.grid(row=1, column=2, padx=5, pady=5)
self.exit_btn = tk.Button(self, text='Exit', command=self.exit_app)
self.exit_btn.grid(row=1, column=3, padx = 5, pady=5)
self.write_to_log('tankfillmodbusapp ')
self.write_to_log(__version__)
self.write_to_log('\n')
self.adapter = get_port()
if self.adapter != '':
self.write_to_log('Port found: ')
self.write_to_log(self.adapter)
self.write_to_log('\n')
self.client = ModbusSerialClient(port=self.adapter, baudrate=BAUDRATE,
bytesize=BYTESIZE, parity=PARITY,
stopbits=STOPBITS, timeout=TIMEOUT,
retry_on_empty=RETRYONEMPTY,
reconnect_delay=RECONNECTDELAY)
self.client.connect()
if self.client.is_socket_open():
self.write_to_log('Connected\n')
self.t = TankFill(TANK_DIAMETER, TANK_HEIGHT, TAU, MAX_IN_FLOW,
TIME_STEP, PLC_MAX_PUMP_SPEED, PLC_MAX_WATER_HEIGHT)
self.do_modbus_comm()
else:
self.write_to_log('Not connected')
self.clear_btn.configure(state='disabled')
self.save_btn.configure(state='disabled')
self.save_exit_btn.configure(state='disabled')
```

```
def write_to_log(self, msg):
self.log['state'] = 'normal'
self.log.insert('end', msg)
self.log['state'] = 'disabled'

def on_clear_data(self):
self.t.clear_sim_data()
self.write_to_log('Data cleared\n')

def on_save_data(self):
self.t.save_sim_data()
self.write_to_log('Data saved: ')
self.write_to_log(self.t.get_last_data_file())
self.write_to_log('\n')

def do_modbus_comm(self):
if not self.client.is_socket_open():
self.write_to_log('Reconnecting\n')
self.client.connect()
if self.exiting == False:
# Send current water height to the PLC
wr = self.client.write_register(address=WATER_HEIGHT_ADDR,
value=self.t.get_plc_water_height(),
slave=UNIT)
# Ignore write errors for now
if self.exiting == False:
# Get the current desired pump speed and the current set point from the PLC
rr = self.client.read_holding_registers(address=PUMP_SPEED_ADDR, count=2,
slave=UNIT)
if not rr.isError():
self.t.set_plc_pump_speed(rr.registers[0])
self.t.set_plc_set_point(rr.registers[1])
if self.exiting == False:
self.after(1, self.do_modbus_comm)

def save_exit_app(self):
if self.t != None:
self.on_save_data()
self.exit_app()

def exit_app(self):
self.exiting = True
if self.client != None and self.client.is_socket_open() == True:
self.client.close()
self.destroy()

if __name__ == "__main__":
root = App()
root.mainloop()
```

The following is the Python source code of the TankFill class that implements the tank fill simulation.

```
'''
TankFill - A simulation of filling a tank with a variable speed pump.

Brad L. Kicklighter, P.E.
4-15-2022
'''

__author__ = "Brad L. Kicklighter, P.E."
__version__ = "2023.01.08"

import math
from unitconv import *
from repeatedtimer import *
from physical_constants import *
import userpaths
import csv

class TankFill(object):
'''
Class to represent a cylindrical tank with variable flow rate input
from a variable speed pump. Water flows out of tank at bottom by
gravity.

Set parameters of system when creating object.
Calling begin() function starts periodic timer that calls
_calc_water_height(). Calling end() ends the simulation.
Call set_plc_pump_speed() to set current pump speed from PLC
and calculate input flow rate. Call get_plc_water_height() to get
water height scaled for PLC. Call set_plc_set_point to set the set
point from the PLC and call get_plc_set_point to get the set
point from the PLC.
'''

####################################################################
#Public Interface
```

```
####################################################################
def __init__(self, tank_diameter, tank_height, tau, max_inflow, delta_t,
  max_plc_pump_speed, max_plc_water_height):
'''
Initialization of TankFill object.

:param tank_diameter: Diameter of tank (m)
:param tank_height: Height of tank (m)
:param tau: Time constant of tank outflow (s)
:param max_inflow: Maximum flow rate of pump (kg/s)
:param delta_t: Simulation time step (s)
:param max_plc_pump_speed: Maximum pump speed from PLC
:param max_plc_water_height: Maximum water height to PLC
'''

# System Properties
self.tank_diameter = tank_diameter  # Tank diameter in meters (m)
self.tank_height = tank_height  # Tank height in meters (m)
self.tank_area = self._area_circle(self.tank_diameter)  # Tank base area in square
  meters (m^2)
self.tank_volume = self._volume_cyl(self.tank_area, self.tank_height)  # Tank volume in
  cubic meters (m^3)
self.tau = tau  # System time constant in seconds (s)
self.max_inflow = max_inflow  # Maximum input flow rate in kilograms per second (kg/s)
self.delta_t = delta_t  # Simulation time step in seconds (s)
self.documents_path = userpaths.get_my_documents()

# PLC Properties
self.max_plc_pump_speed = max_plc_pump_speed  # Maximum pump speed from PLC
self.max_plc_water_height = max_plc_water_height  # Maximum water height to PLC

# Current Values
self.inflow = 0.0  # Current input flow rate in kilograms per second (kg/s) (0.0 to
  max_inflow)
self.water_height = 0.0  # Current height of water in meters (m) (0.0 to tank_height)
self.plc_water_height = 0  # Current height of water scaled for PLC (integer 0 to
  max_plc_water_height)
self.plc_pump_speed = 0  # Current pump speed from PLC (integer 0 to max_plc_pump_speed)
self.plc_set_point = 0  # Current set point (not used by model, just part of data
  collection)

# Simulation data
self.sim_data = []
self._append_sim_data_header()
self.start_time = time.perf_counter()  # Get start time of simulation
self.last_data_file = ''  # Path and name of last data file saved

# Timer Thread
self.timer = RepeatedTimer(self.delta_t, self._calc_water_height)

def begin(self):
'''
Start tank simulation.
'''
self.timer.start()  # RepeatedTimer self starts so this is not necessary

def end(self):
'''
End tank simulation.
'''
self.timer.stop()

def set_plc_pump_speed(self, plc_pump_speed):
'''
Clamps PLC pump speed, stores value, and calculates input flow rate.

:param plc_pump_speed: pump speed from PLC (0 to max_plc_pump_speed)
'''
self.plc_pump_speed = max(min(plc_pump_speed, self.max_plc_pump_speed), 0)
self._calc_inflow()

def get_plc_pump_speed(self):
'''
Returns the current pump speed from the PLC.

:returns: pump speed (0 to max_plc_pump_speed)
'''
return self.plc_pump_speed

def get_plc_water_height(self):
'''
Returns the current water height scaled for PLC.

:returns: water height (0 to max_plc_water_height)
'''
return self.plc_water_height

def get_plc_set_point(self):
'''
Returns the current set point from the PLC.
```

```
:returns: set point
'''
return self.plc_set_point

def set_plc_set_point(self, set_point):
'''
Sets the current set point from the PLC.
'''
self.plc_set_point = set_point

def clear_sim_data(self):
'''
Clears simulation data.
'''
self.sim_data.clear()
self._append_sim_data_header()
self.start_time = time.perf_counter()

def save_sim_data(self):
'''
Save simulation data in documents folder with file name containing current date and
time. Simulation data is cleared after save.
'''
self.last_data_file = self.documents_path + "\\" + "simulation_data_" +
  time.strftime("%Y%m%d-%H%M%S") + ".csv"
with open(self.last_data_file, 'w', newline='') as csvfile:
csvwriter = csv.writer(csvfile)
csvwriter.writerows(self.sim_data)
self.clear_sim_data()

def get_last_data_file(self):
'''
Gets path and name of last data file saved.

:returns: path and name of last data file saved
'''
return self.last_data_file

####################################################################
#Private Interface
####################################################################
def _clamp_height(self, val):
'''
Clamps a water height in meters (m) to 0.0 to tank_height.
```

```
:param val: water height (m)
:returns: clamped height (m)
'''
return max(min(val, self.tank_height), 0.0)

# TODO: move area and volume functions to their own file
def _area_circle(self, diameter):
'''
Calculates the area of a circle.

:param diameter: diameter of circle
:returns: area of circle
'''
return math.pi * (diameter / 2) ** 2

def _volume_cyl(self, area, height):
'''
Calculates the volume of a cylinder.

:param area: area of cylinder base
:param height: height of cylinder
:returns: volume of cylinder
'''
return area * height

def _calc_water_height(self):
'''
Calculates current water height in meters (m) and scaled for PLC. Height is clamped.
Appends simulation data to list.
'''
self.water_height = self._clamp_height(self.water_height + self.delta_t * (self.inflow
  / (DENSITY_WATER * self.tank_area) - self.water_height / self.tau))
self._calc_plc_water_height()
self._append_sim_data()

def _calc_plc_water_height(self):
'''
Calculates the water height as an integer scaled for the PLC.
'''
self.plc_water_height = int(round(self.max_plc_water_height * self.water_height /
  self.tank_height))

def _calc_pump_mtr_frac(self):
'''
```

```
Calculates the pump motor fraction based on the pump speed from the PLC.

:returns: pump motor drive fraction (0.0 to 1.0)
'''
return float(self.plc_pump_speed) / float(self.max_plc_pump_speed)

def _calc_inflow(self):
'''
Calculates current input flow rate in kilograms per second (kg/s) based on pump motor
percentage.

:returns: input flow rate (kg/s)
'''
self.inflow = self._calc_pump_mtr_frac() * self.max_inflow
return self.inflow

def _calc_outflow(self):
'''
Calculates current output flow rate in kilograms per second (kg/s).

:returns: output flow rate (kg/s)
'''
return DENSITY_WATER * self.water_height * self.tank_area / self.tau

def _append_sim_data(self):
'''
Appends current time, PLC pump speed, PLC water height, and PLC set point to simulation
data.
'''
self.sim_data.append([time.perf_counter() - self.start_time,
self.get_plc_pump_speed(),
self.get_plc_water_height(),
self.get_plc_set_point()])

def _append_sim_data_header(self):
'''
Appends header record to simulation data.
'''
self.sim_data.append(["Time (s)", "Pump Speed", "Water Height",
"Set Point"])
```