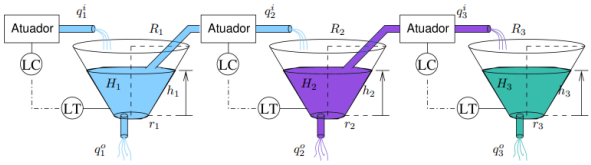
TRABALHO FINAL
PROCESSO DE CONTROLE DE TANQUES

Marcela Fontes Abreu – 2018013798

Descrição do problema

O trabalho foi desenvolvido com o objetivo de consolidar alguns dos principais temas abordados ao longo do semestre. Entre os tópicos estudados estão comunicações OPC e TCP/IP, modelagem de sistemas e técnicas de controle.

O sistema escolhido para aplicação desses conceitos foi o controle de nível de tres tanque, conforme ilustrado na figura abaixo:



Com as seguintes equações:

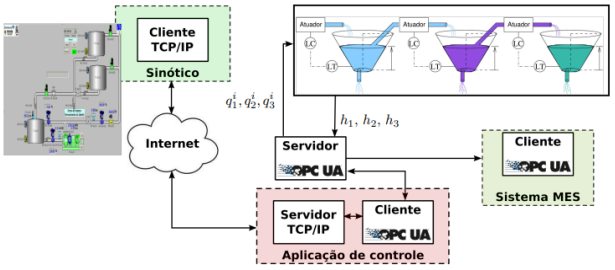
$$q_i^o(t) = \gamma_i \sqrt{h_i(t)}, \quad \text{com} \quad i = 1, 2, 3$$

$$\dot{h}_1(t) = \frac{q_1^i(t) - q_1^o(t) - q_2^i(t)}{\pi \left[r_1 + \frac{R_1 - r_1}{H_1} h_1(t) \right]^2},$$

$$\dot{h}_2(t) = \frac{q_2^i(t) - q_2^o(t) - q_3^i(t)}{\pi \left[r_2 + \frac{R_2 - r_2}{H_2} h_2(t) \right]^2}$$

$$\dot{h}_3(t) = \frac{q_3^i(t) - q_3^o(t)}{\pi \left[r_3 + \frac{R_3 - r_3}{H_3} h_3(t) \right]^2}.$$

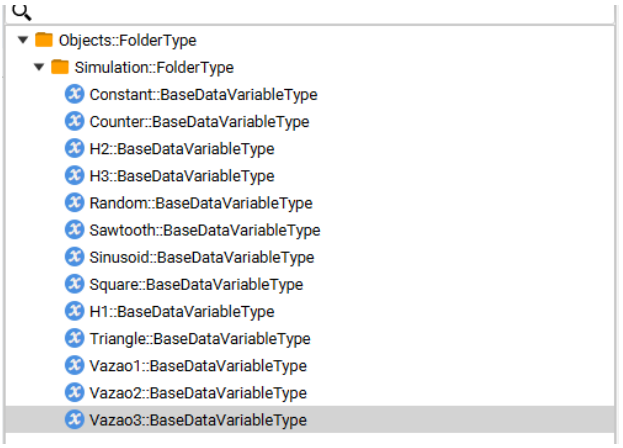
Utilizando a seguinte estrutura de sistema distribuído:



Criação do Cliente OPC

Variáveis criadas para nodes das variáveis.

Para que o programa seja corretamente executado é necessário inicializar o servidor OPC Prosys OPC UA Simulation Server e adicionar as variáveis de altura do líquido de cada tanque, ex.: ("ns=3;i=1008") e vazão de entrada ("ns=3;i=1009").



CLP.py

O programa CLP é composto por 2 threads, uma contém o cliente OPC UA, que vai ler as informações e com elas atuar no processo, e a outra contém o servidor TCP/IP, responsável por receber as informações do cliente TCP/IP do sinóptico. A altura dos tanques, h1, h2 e h3, é uma variável global, pois será usada em ambas as threads. Para que o controle do sistema seja feito, é necessária uma altura de referência, essa altura é definida pelo usuário e enviada do cliente TCP/IP para o servidor no programa CLP, esse valor é compartilhado com o cliente OPC UA, onde o processo de controle é feito. O servidor TCP envia para o seu cliente, programa tcp_client descrito no próximo item, os dados de altura em tempo real, esses dados serão usados para a construção da interface gráfica do trabalho. O controle do processo foi feito através de um controlador PID, o controlador atua na altura um que por sua vez atua nas outras vazoes etc, implementado na função control_system, e as suas constantes foram determinadas experimentalmente. Esse controle só começa a agir quando o cliente passa o valor de referência, ou seja, o nível desejado para os tanques.

```
class ClientOPC(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.qin1 = 0.0
        self.qin2 = 0.0
        self.qin3 = 0.0
        self.I = 0.0
        self.error = 0.0
        self.prev_error = 0.0

    def run(self):
        clientOPC = Client("opc.tcp://localhost:53530/OPCUA/SimulationServer")

        clientOPC.connect()

        node1 = clientOPC.get_node("ns=3;i=1008")
```

```
class ServerTCP(Thread):
    def __init__(self):
        Thread.__init__(self)

    def run(self):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
        s.bind((socket.gethostname(), 8000))
        s.listen()
        clientsocket, address = s.accept()

        while True:
            data = json.loads(clientsocket.recv(1024))
            if not data:
                break
            global href
            href = float(data)
```

tcp_cliente.py

O programa tcp_client é responsável por enviar dados remotamente para o processo, através de uma comunicação cliente/servidor TCP/IP, sendo este programa o cliente.

Fazendo um paralelo com uma aplicação real, é como se este programa fosse o sinótico do sistema, em uma sala de controle, separada do processo, com comunicação via
internet. Ele permite que o usuário insira o valor desejado para o nível dos tanques, enviando este valor para o servidor TCP/IP no programa clp, descrito anteriormente.

Após enviar para o programa clp o valor do nível de referência, o cliente recebe do
servidor indicando o nível atual do processo e anotando-o em um arquivo .txt denominado "historiador", esses dados são demonstrados tanto no arquivo "historiador", quanto no terminal.

Além disso, o tcp_client escreve os valores de vindos do servidor em um arquivo MES.txt. Quando a conexão do servidor é encerrada, o cliente também se encerra.

```
def tcp_client(href):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((socket.gethostname(), 8000))
    start_time = time.time()
    sending_data = True

    while sending_data:
        s.sendall(bytes(json.dumps(href).encode()))
        sending_data = False

    while True:
        try:
            data_rcv = json.loads(s.recv(1024))
            f = open("historiador.txt", "a")
            f.write(data_rcv + "\n")
            if float(data_rcv)/5.0 < 0.05:
                with open('below.txt', 'r') as f:
                    for line in f:
                        print(line.rstrip())

            elif float(data_rcv)/5.0 > 0.95:
                with open('above.txt', 'r') as f:
                    for line in f:
                        print(line.rstrip())

            graph_data = open("MES.txt", "a")
            graph_data.write(str(round(time.time()-start_time))+","+str(data_rcv)+"\n")
        except:
```

tanque.py

O arquivo que contém os cálculos de altura e de vazão de saída:

```
def tcp_client(href):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((socket.gethostname(), 8000))
    start_time = time.time()
    sending_data = True

    while sending_data:
        s.sendall(bytes(json.dumps(href).encode()))
        sending_data = False

    while True:
        try:
            data_rcv = json.loads(s.recv(1024))
            f = open("historiador.txt", "a")
            f.write(data_rcv + "\n")
            if float(data_rcv)/5.0 < 0.05:
                with open('below.txt', 'r') as f:
                    for line in f:
                        print(line.rstrip())

            elif float(data_rcv)/5.0 > 0.95:
                with open('above.txt', 'r') as f:
                    for line in f:
                        print(line.rstrip())

            graph_data = open("MES.txt", "a")
            graph_data.write(str(round(time.time()-start_time))+","+str(data_rcv)+"\n")
        except:
```

```
def tank1_behavior():
        self.h = (qin1-y1*math.sqrt(self.h1)-qin2)/(math.pi*(r1+alpha*self.h1)^2)
        print("Altura do líquido do tanque 1: ", self.h1)
        self.qin1 = y1*math.sqrt(self.h1);
        node1.set_value(self.h1)
        node2.set_value(self.qin1)
def tank2_behavior():
        self.h = (qin2-y2*math.sqrt(self.h2)-qin3)/(math.pi*(r1+alpha*self.h2)^2)
        print("Altura do líquido do tanque 2: ", self.h2)
        self.qin2 = y2*math.sqrt(self.h2);
        node3.set_value(self.h2)
        node4.set_value(self.qin2)
def tank3_behavior():
        self.h = (qin3-y3*math.sqrt(self.h3))/(math.pi*(r1+alpha*self.h3)^2)
        print("Altura do líquido do tanque 3: ", self.h3)
        self.qin2 = y3*math.sqrt(self.h3);
        node5.set_value(self.h3)
        node6.set_value(self.qin3)
```

main.py

O código main tem como principal função garantir que todos os componentes do sistema funcionem de maneira integrada e correta, atendendo aos objetivos propostos pelo trabalho. Ele é responsável por inicializar e organizar as threads e processos necessários para o funcionamento do sistema.

A função animate é responsável por criar a interface gráfica do projeto, gerando um gráfico que mostra o nível do líquido no tanque 1 em função do tempo. Para isso, ela lê os dados armazenados no arquivo MES.txt, que é preenchido pelo tcp_client com as informações do nível do líquido ao longo do tempo. O arquivo MES.txt é resetado a cada execução do programa, garantindo que os dados antigos não interfiram nos novos resultados.

No main, as threads descritas anteriormente (como as responsáveis pela comunicação OPC e TCP/IP) são inicializadas e organizadas conforme as especificações do trabalho. Ao final da execução, essas threads são encerradas de forma adequada, assim como o programa como um todo. Além disso, a função main inicia a função animate, permitindo que o gráfico seja gerado e exibido em tempo real, proporcionando uma visualização clara do comportamento do nível do líquido nos tanques.

```python
def main():
    open('MES.txt', 'w').close()

    client_opc_thread = ClientOPC()
    server_tcp_thread = ServerTCP()
    tanque_conico_thread = TanqueConico()

    href = input("Insira o setpoint de altura do tanque (Deve ser menor que a altura má

    tcp_client_process = Process(target=tcp_client, args=[href])

    server_tcp_thread.start()
    tcp_client_process.start()
    tanque_conico_thread.start()
    client_opc_thread.start()

    ani = animation.FuncAnimation(fig, animate)
    plt.show()

    client_opc_thread.join()
    tanque_conico_thread.join()
    server_tcp_thread.join()
    tcp_client_process.terminate()

    print("Fim do ciclo de controle")
```

timer.py

O timer foi desenvolvido para possibilitar a execução de tarefas periódicas no projeto. Ele é responsável por garantir que ações específicas sejam realizadas em intervalos de tempo definidos. Por exemplo, o controlador atua a cada 0,2 segundos, enquanto a logger_thread registra os valores a cada 1 segundo, entre outras tarefas que dependem de repetição em intervalos regulares. Essa funcionalidade é essencial para o funcionamento sincronizado e eficiente do sistema.

```python
from threading import Timer

class LoopTimer(Timer):
    def run(self):
        while not self.finished.wait(self.interval):
            self.function(*self.args, **self.kwargs)
```

Problemas não Solucionados

Controle não funciona da maneira como deveria ser na prática, devido à maneira de como são 3 tanques e necessita de maior controle por depender de 3 vazões de saída mais as vazões que tiram do tanque 1.