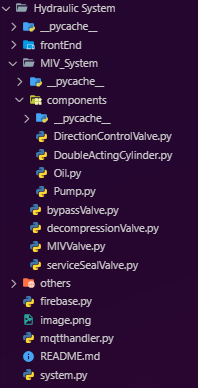
**Folder structure (Using the OKR method)**

****

*Figure 1. Folder structure of MIV system*

The core simulation logic resides within the Hydraulic System directory. This directory further contains subdirectories for specific systems being modeled. Currently, the only subdirectory present is the MIV System subdirectory.

**MIV System Subdirectory:**

This subdirectory encompasses all the necessary logic and simulation model code for simulating the opening and closing sequence of the MIV System. It also includes individual component models that contribute to the overall opening and closing behavior.

**Key Components:**

***system.py:*** This Python file serves as the main entry point for the simulation. It contains the logic responsible for calling the required functions to execute the model.

***Component-specific Python files:*** Individual files like BypassValve.py, DecompressionValve.py, and others likely models the behavior of specific components within the MIV System. These files interact with each other and system.py to create the entire simulation.

**Inside MIV system directory**

1. BypassValve.py

The provided code defines a Bypass Valve class responsible for managing the operation of a bypass valve system. It initializes various parameters related to the valve's state and functionality. The run method of this class runs the simulation process by simulating the operation of different components involved in the system:

To make the bypass Valve a functional working model, it is linked to three .py model filed inside of the component’s directory. Now those individual files in components directory namely:

1. Hydraulic pump
2. Directional Control valve
3. Hydraulic Actuator

Will consist of a mathematical model which calculates the required parameters based on the input parameters provided by the user. (Later it is expected to make a front-end panel where input parameters can be easily set by user based on their needs).

***Hydraulic Pump:*** Simulates the pump's operation based on specified parameters such as *oil type, motor speed, operating pressure, mass, and efficiency.*

***Direction Control Valve:*** Simulates the valve's operation based on *density, viscosity, pump pressure, and flow rate.*

***Hydraulic Actuator:*** Simulates the behavior of the hydraulic actuator based on parameters such as *bore diameter, rod diameter, stroke length, initial position, simulation time, operating pressure, density, discharge, packing friction, and timing.*

The run method coordinates the simulation of these components, updates the system's state, and publishes the simulation data via MQTT for monitoring and analysis. Additionally, it handles the initialization of values and updates the system's time during the simulation process.

def run(self, time\_counter, event\_time, activation):

*#test for hydraulic double acting actuator*

        bore\_diameter = 0.15 *# m*

        rod\_diameter = 0.016 *# mcls*

        stroke\_length = 2 *# m maximum length the pistion rod extends*

        initial\_position = 0 *# m initial position of the piston*

        simulation\_time = 0.25 *# s*

        discharge = 0.0032 *# m^3/s*

        packingFriction = 0 *# N*

        timer = time\_counter

        motor\_speed = 3600

        oil\_type = "skydrol\_1"

        operating\_pressure = 7845320 *# in pascals*

        efficiency = 0.85

        mass =  50000 *# in KG*

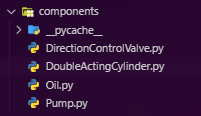
        pump\_ON = True

The above parameters are currently used for simulating the Bypass Valve opening and Closure model. Later those parameters can be changed based on the requirements of the system. Similarly, MIV, Service Seal valve and Decompression valve will be ultimately linked to the components directory models to simulate their working but with different parameters they will give different behavior when we simulate the system.

**How Valves are Working:**

The working of this Valves is possible due to the components or the parts which are there in the component’s directory. In this there are three python files namely DirectionControlValve.py, DoubleActingCylinder.py, and pump.py. Each of the system are modelled to replicate the functionality of the Cylinder piston and Direction control of the fluid. The pump on the other hand is modelled to act like an actual hydraulic pump. But since the parameters are all currently estimation, later we can configure with the real data from the Hydro power plant.

Now each of the valve class will call the components in sequency to get the modelled output in real time basis. The parameters will be set in the valve class itself, all it does is feed those parameters into the model’s to give the desired output.



*Figure 2. Components folder files consisting of the actual mathematical models*

**Double Acting Cylinder file Logic**

This code is responsible for calculating all the parameters, it uses basic physics formula to calculate the require Force and velocity of piston on the real time basis. The model is also responsible for calculating the hydraulic power generated by the piston. The efficiency is the main factor which separates the simulation and real-world phenomena in most cases. Therefore, efficiency factor is considered to give the output of simulation as real as possible.

It takes the below parameters:

        self.**bore\_diameter** = bore\_diameter

        self.**rod\_diameter** = rod\_diameter

        self.**stroke\_length** = stroke\_length

        self.**initial\_position** = initial\_position

        self.**simulation\_time** = simulation\_time

        self.**operating\_pressure** = operating\_pressure

        self.**oil\_flow\_pressure** = oil\_flow\_pressure

        self.**flow\_rate** = flow\_rate

        self.**packingFriction** = packingFriction

        self.**time\_instant** = time\_instant

        self.**event\_time\_instant** = event\_time\_instant

        self.**current\_stroke\_position** = current\_stroke\_position

        self.**signal\_flag\_ext** = signal\_flag\_ext

        self.**last\_stroke\_position** = last\_stroke\_position

        self.**signal\_flag\_ret** = signal\_flag\_ret

        self.**position\_data** = {}

        self.simulate = self.simulate(port)

To give the following output

# this is for the extension of piston

            f\_extension = self.extensionForce()

            v\_extension = self.pistonVelocityExt()

            power\_input = self.powerInputExt()

            power\_output= self.powerOutputExt()

            displacement = self.displacementExt(self.**event\_time\_instant**)

            q = self.massFlowRateExt()

# this is the retraction of the cylinder

            f\_retraction = self.retractionForce()

            v\_retraction = self.pistonVelocityRet()

            power\_input = self.powerInputRet()

            power\_output = self.powerOutputRet()

            displacement = self.displacementRet(self.**event\_time\_instant**)

Now considering the factor such as force, velocity, power and displacement of the double acting cylinder, it can behave like a real cylinder on real time basis considering the efficiency factor Aswell.

Formulas used to model the hydraulic system.

    def extensionForce(self):

        force\_extension = self.**operating\_pressure** \*(np.**pi** \* pow(self.**bore\_diameter**/2, 2)) *#in newtons*

        return force\_extension

    def massFlowRateExt(self):

*# massflow rate (m) = PAV*

        mass\_flow\_rate = self.**oil\_flow\_pressure** \* self.**flow\_rate**

*# print("Massflr = " + str(mass\_flow\_rate) + " kg/s")*

        return mass\_flow\_rate

*# important function to determine the postion of the extension  piston in meters*

    def pistonVelocityExt(self):

        if (self.**signal\_flag\_ext**):

*# velocity = self.stroke\_length / self.simulation\_time #just calculating*

            velocity = self.**flow\_rate** / (np.**pi** \* (pow(self.**bore\_diameter**/2, 2)))

            if (self.**current\_stroke\_position** >= self.**stroke\_length**):

                velocity = 0

        else:

            velocity = 0

        return velocity

    def powerInputExt(self):

        power\_in = self.**operating\_pressure** \* self.**flow\_rate**

        return power\_in

    def powerOutputExt(self):

*#  after considerring packing friction*

        forecenet = self.**operating\_pressure** \* (np.**pi** \* pow(self.**bore\_diameter**/2,2 )) - self.**packingFriction**

        p\_out = forecenet \* self.pistonVelocityExt()

        return p\_out

    def pistonEfficiencyExt(self):

        efficiency = self.powerOutputExt() / self.powerInputExt()

        return efficiency

    def pistonAccelerationExt(self):

        acceleration = self.pistonVelocityExt() / self.**simulation\_time**

        return acceleration

    def retractionForce(self):

        force\_retraction = self.**operating\_pressure** \* (np.**pi**\*(pow(self.**bore\_diameter**/2, 2) - pow(self.**rod\_diameter**/2, 2)))

*# print("retraction force = "+ str(force\_retraction)+" N")*

        return force\_retraction

*# important  function as it defines the postion of the piston head during the retraction*

    def pistonVelocityRet(self):

        if (self.**signal\_flag\_ret**):

            velocity = self.**flow\_rate** / ((np.**pi** \* pow(self.**bore\_diameter**/2, 2))-(np.**pi** \* pow(self.**rod\_diameter**/2, 2))) *# m/s*

*# print("return velocity ---------->", velocity)*

            if (self.**current\_stroke\_position** <= 0):

                velocity = 0

        else:

            velocity = 0

        return velocity

    def powerInputRet(self):

        power\_in = self.**operating\_pressure** \* self.**flow\_rate**

        return power\_in

    def powerOutputRet(self):

        netForce = (self.**operating\_pressure** \* ((np.**pi**\*pow(self.**bore\_diameter**/2,2))-(np.**pi** \* pow(self.**rod\_diameter**/2,2)))) - self.**packingFriction**

        power\_out = netForce \* self.pistonVelocityRet()

        return power\_out

    def pistonEfficiencyRet(self):

        eff = self.powerOutputRet()/ self.powerInputRet()

        return eff

*# calculating the position of th top of the pistion during the extension phase*

    def displacementExt(self, time\_instant):

        position = self.**current\_stroke\_position**

*# print("position ---> ", position)*

        if ( position >= 0 and position < self.**stroke\_length**):

            if (self.**signal\_flag\_ext**):

                disp = self.**last\_stroke\_position** + (self.pistonVelocityExt() \* time\_instant)

            else:

                disp = self.pistonVelocityExt() \* time\_instant

            return disp

        elif (position >= self.**stroke\_length**):

            disp = self.**stroke\_length**

            return disp

    def displacementRet(self, time\_instant):

        if (self.**current\_stroke\_position** <= 0):

*# print("im hre")*

            disp = 0

            return disp

        elif(self.**current\_stroke\_position** > 0):

*# print("im here")*

            if (self.**signal\_flag\_ret** and self.**current\_stroke\_position** < self.**stroke\_length**):

                disp = self.**last\_stroke\_position** - (self.pistonVelocityRet() \* time\_instant)

            else:

                disp = self.**stroke\_length** - (self.pistonVelocityRet() \* time\_instant)

*# print("disp--->", disp)*

            return disp

**Pump Logic:**

Pump on the other hand takes the different oil and their properties to calculate the pressure of the liquid flowing and the flow rate.

The input parameters which can be changes by the user

pump\_type,

motor\_speed,

operating\_pressure,

load\_mass,

pump\_efficiency\_factor

self.**acceleration\_gravity** = 9.81 *# m/s2'*

self.**density**, self.**viscosity**, self.**bulk\_modulus** = HydraulicOil(pump\_type).**fluid\_properties**