run_experiment

January 15, 2024

In [ ]:
```
import datetime
from pathlib import Path
import asyncio
import time
from loguru import logger
```

In [ ]:
```
from BV_experiments.platform_error import PlatformError
```

In [ ]:
```
# Flowchem devices
from flowchem.client.client import get_all_flowchem_devices

flowchem_devices = get_all_flowchem_devices()

# pressure
press_control = flowchem_devices['pressEPC']['EPC']
press_helper = flowchem_devices['pressMFC']['MFC']

# fill the loop
loop_pump = flowchem_devices["syr5"]["pump"]
loop_valve = flowchem_devices['r2']['InjectionValve_A']

# reation setup
deliver_gas = flowchem_devices['O2MFC']['MFC']
deliver_liquid = flowchem_devices['Knauer-pumpM']['pump']
# deliver_liquid = flowchem_devices["r2"]['Pump_A']
reactor_temp = flowchem_devices["r2"]["reactor-3"]
reactor_photo = flowchem_devices["r2"]['PhotoReactor']

# collect reaction mixture
collect_valve = flowchem_devices["r2"]['CollectionValve']
wash_liquid = flowchem_devices["r2"]['Pump_A']
# wash_liquid = flowchem_devices['Knauer-pumpM']['pump']

# transfer reaction mixture
transfer_liquid = flowchem_devices["ML600"]["left_pump"]
transfer_valve = flowchem_devices["ML600"]['left_valve']
# transfer_liquid = flowchem_devices["syr3"]["pump"]
```

```
        # transfer_valve = flowchem_devices['6PortValve']['distribution-valve']

        # analysis
        dilute_liquid = flowchem_devices["r2"]['Pump_B']
        hplc_valve = flowchem_devices['HPLCvalve']['injection-valve']

        # power
        r2_power = flowchem_devices['r2']['Power']
        bubble_power = flowchem_devices['bubble-sensor-power']['5V']

        # sensor
        general_exp_sensor = flowchem_devices['r2']['GSensor2']
        pumpM_pressure = flowchem_devices['Knauer-pumpM']['pressure']
        bubble = flowchem_devices['bubble-sensor-measure']['bubble-sensor']
```

In [ ]:
```
        # 6PortValve
        tranfer_valve_mapping = {
            "pump": "transfer", "1": "vial", "2": "waste",
            "3": None, "4": None, "5": None, "6": "analysis"}

        # ML600
        tranfer_valve_mapping = {
            "syr-right": "vial", "syr-front": "analysis", "syr-left": "waste"}

        r_valve_mapping = {v: k for k, v in tranfer_valve_mapping.items()}
```

In [ ]:
```
        async def exp_hardware_init():
            logger.info("____ initialize all hardware ____")
            # reactor
            reactor_photo.put("power-off")
            reactor_temp.put("temperature", params={
                "temperature": f"22řC", "heating": "true", })

            # gas/syringe/hplc pump
            deliver_gas.put("stop")
            loop_pump.put("stop")
            deliver_liquid.put("stop")
            transfer_liquid.put("stop")
            dilute_liquid.put("stop")
            wash_liquid.put("stop")

            loop_valve.put("position", params={"position": "load"})
            collect_valve.put("position", params={"position": "Solvent"})
            transfer_valve.put("position", params={
                "position": r_valve_mapping["waste"]})

            hplc_valve.put("position", params={"position": "load"})
            # power
```

```
            r2_power.put("power-off")
            bubble_power.put("power-off")
            bubble.put("power-off")

            press_control.put("stop")
            press_helper.put("stop")
```

```
In [ ]: async def transfer(
                withdraw_p: str,
                infuse_p: str,
                withdraw_vol: float = 0.25,
                infuse_vol: float = 0.25,
                withdraw_speed: float = 1.0,
                infuse_speed: float = 1.0,
                max_transfer_vol: float,
                wait_to_finish_infuse: bool = True,
                viscosity: bool = True  # todo: to test
        ):
            """
            transfer single unit (per syringe) without change flow rate

            :param withdraw_p: withdraw position
            :param infuse_p: infuse position
            :param withdraw_speed: flow rate to transfer solution (default 1.0 ml/min)
            :param infuse_speed: flow rate to transfer solution (default 1.0 ml/min)
            :param withdraw_vol: should the maximum transfer volume
            :param infuse_vol:
            :param max_transfer_vol: define the maximum transfer each time
            :param wait_to_finish_infuse: wait to finish to start next step
            :param viscosity:

            """
            # check the volume is doable
            if withdraw_vol > max_transfer_vol or infuse_vol > max_transfer_vol:
                raise PlatformError(f"the max transfer vol only {max_transfer_vol} ml."
                                    f"Check required transfer volume")

            # real transfer
            logger.info("____ one transfer ____")
            # withdraw
            logger.debug("withdraw")
            transfer_valve.put("position", params={"position": withdraw_p})

            withdraw_time = withdraw_vol / withdraw_speed
            transfer_liquid.put("withdraw", params={
                "rate": f"{withdraw_speed} ml/min", "volume": f"{withdraw_vol} ml"})
            await asyncio.sleep(withdraw_time * 60)
            await asyncio.sleep(2) if viscosity else None
```

```
            # infuse
            logger.debug("infuse")
            transfer_valve.put("position", params={"position": infuse_p})

            infuse_time = infuse_vol / infuse_speed
            transfer_liquid.put("infuse", params={
                "rate": f"{infuse_speed} ml/min", "volume": f"{infuse_vol} ml"})
            await asyncio.sleep(infuse_time * 60) if wait_to_finish_infuse else None
```

In [ ]:
```
async def deliver_specific_vol(
        volume: float,
        last_full_withdraw: bool,
        withdraw_p: str, infuse_p: str,
        withdraw_spd: float = 1.0, infuse_spd: float = 1.0,
        max_transfer_vol: float | None = None,
        wait_to_finish_infuse: bool = False):
    """
    the function is used to deliver set volume
    :param volume: deliver total volume in ml
    :param last_full_withdraw: control the last withdraw is full syringe
                    or rest volume
    :param withdraw_p: control valve position (input)
    :param infuse_p: deliver valve position (output)
    :param withdraw_spd: withdraw speed (ml/min)
    :param infuse_spd: infuse speed (ml/min)
    :param max_transfer_vol: define the maximum transfer each time
    :param wait_to_finish_infuse:

    :return:
    last_w_vol-volume: provide info of rest volume in syringe
    """
    from math import floor
    full_transfer_n = floor(volume / max_transfer_vol)  #
    last_transfer_vol = round(volume % max_transfer_vol, 10)

    for i in range(full_transfer_n):
        logger.debug(f"The {i + 1} time of transfer. "
                     f"Still {full_transfer_n - i - 1} times of full transfer.")
        await transfer(withdraw_p=withdraw_p, infuse_p=infuse_p,
                       withdraw_vol=max_transfer_vol,
                       infuse_vol=max_transfer_vol,
                       withdraw_speed=withdraw_spd,
                       infuse_speed=infuse_spd,
                       wait_to_finish_infuse=True, viscosity=False)
        volume -= max_transfer_vol

    if last_transfer_vol == 0:
```

```
            return 0

        # last deliver
        logger.debug(f"The last time of transfer.")
        last_w_vol = max_transfer_vol if last_full_withdraw else last_transfer_vol
        await transfer(withdraw_p=withdraw_p, infuse_p=infuse_p,
                       withdraw_vol=last_w_vol, infuse_vol=last_transfer_vol,
                       withdraw_speed=withdraw_spd, infuse_speed=infuse_spd,
                       wait_to_finish_infuse=wait_to_finish_infuse, viscosity=False,
                       )
        logger.info(f"still {last_w_vol - last_transfer_vol} ml left in syringe.")
        return last_w_vol - last_transfer_vol
```

```
In [ ]: async def run_experiment(condition: dict,
                                 calculator):

        flow_rate = calculator.calc_gas_liquid_flow_rate(condition)
        prep_sys_para = calculator.calc_stable_system(condition, gl_flow=flow_rate)
        schedule = calculator.reaction_schedule(condition, gl_flow=flow_rate)

        r2_power.put("power-on")

        # pre-run
        deliver_gas.put("set-flow-rate", params={"flowrate": f"{prep_sys_para['pre_gas_flow
        deliver_liquid.put("infuse", params={
            "rate": f"{prep_sys_para['pre_liquid_flow']} ml/min"})  # fixme:flowrate vs sp

        reactor_photo.put("intensity", params={"percent": f"100"})
        reactor_temp.put("temperature", params={"temperature": f"{condition['temperature']
                                                "heating": "false",
                                                })
        await asyncio.sleep(schedule['pre_run_time'] * 60)

        deliver_gas.put("set-flow-rate", params={
            "flowrate": f"{flow_rate['gas_flow']} ml/min"})
        deliver_liquid.put("infuse", params={"rate": f"{flow_rate['liquid_flow']} ml/min"}

        #  fill the loop
        loop_pump.put("infuse", params={"rate": "1.0 ml/min", "volume": "1.0 ml"})
        await asyncio.sleep(schedule["fill_loop"] * 60)

        # check system
        logger.info("____ start check the system is ready or not ____")
        from BV_experiments.Example0_BV.platform_individual import check_system_ready
        sys_state = await check_system_ready(condition, flow_rate['gas_flow'], 20.0)

        if not sys_state:
            logger.error("Platform could not reach the target condition...")
```

```
            await exp_hardware_init()
            raise PlatformError("Platform could not reach the target condition...")

        # RUN experiment :wait till reaction mixture come out
        loop_valve.put("position", params={"position": "inject"})
        await asyncio.sleep(schedule["loop_to_vial"] * 60)

        # collect the reaction mixture
        collect_valve.put("position", params={"position": "Reagent"})
        logger.info(f"start to collect the reaction mixture.")
        await asyncio.sleep(schedule["collect_time"] * 60)
        collect_valve.put("position", params={"position": "Solvent"})

        coll_vol = schedule["collect_time"] * flow_rate["liquid_flow"]
        logger.info(f"end collecting reaction mixture. total collected volume: {coll_vol} m
        return coll_vol
```

In [ ]:
```
async def analyze_experiment(exp_id,
                             condition,
                             calculator,
                             hplc_commander):
    logger.info("____ start analysis process ____")

    col_vol, col_conc = calculator.vial_sol_info(condition)
    syr_delivered_rate = 0.2
    make_up_flow, final_flow = calculator.calc_dilute_flow(col_conc,
                                                           final_conc=0.01,
                                                           delivered_rate=syr_delivered
    # get rid of air from the tube
    viscosity = True
    [await transfer(withdraw_p="1", infuse_p="2", withdraw_vol=0.05, infuse_vol=0.05,
                    withdraw_speed=1, infuse_speed=1,
                    wait_to_finish_infuse=True, viscosity=viscosity) for x in range(2)]
    logger.debug("finish twice de-bubble process (0.05 ml each)!")

    # withdraw from vial
    transfer_volume = 1.0
    transfer_valve.put("position", params={"position": r_valve_mapping["vial"]})
    transfer_liquid.put("withdraw", params={
        "rate": "1.0 ml/min", "volume": f"{transfer_volume} ml"})
    await asyncio.sleep(1.0 * 60)
    await asyncio.sleep(2) if viscosity else None

    # infuse
    transfer_valve.put("position", params={"position": r_valve_mapping["analysis"]})
    transfer_liquid.put("infuse", params={
        "rate": f"{syr_delivered_rate} ml/min", "volume": f"{transfer_volume} ml"})
    total_infuse_time = transfer_volume / syr_delivered_rate
```

```
            await asyncio.sleep(total_infuse_time * 0.1 * 60)
            dilute_liquid.put("infuse", params={"rate": f"{make_up_flow} ml/min"})
            await asyncio.sleep(total_infuse_time * 0.4 * 60)

            # submit the hplc
            await hplc_commander.load_method(r"methionine_method_10min.MET")
            await hplc_commander.set_sample_name(f"{exp_id}")
            await hplc_commander.run()  # delay 2 sec.....
            await asyncio.sleep(2)
            hplc_valve.put("position", params={"position": "inject"})

            logger.info(f"Switch the hplc injection valve and start to analysis")

            # empty the syringe
            dilute_liquid.put("stop")
            await asyncio.sleep(total_infuse_time * 0.5 * 60)
            logger.info(f"finish emptying {transfer_volume} ml reaction mixture.")
            return col_vol - 1.0
```

In [ ]:
```
async def wash_system(left_volume):
            logger.info(f"____ empty vial ____")
            await deliver_specific_vol(volume=left_volume, last_full_withdraw=False,
                                       withdraw_p=r_valve_mapping["vial"],
                                       infuse_p=r_valve_mapping["waste"],
                                       withdraw_spd=1.0, infuse_spd=1.0,
                                       max_transfer_vol=1.0,
                                       wait_to_finish_infuse=True)

            logger.info(f"____ rinse vial 2 times____")
            # fill the vial
            collect_valve.put("position", params={"position": "Reagent"})
            wash_liquid.put("infuse", params={"rate": f"{5} ml/min"})
            await asyncio.sleep(3 / 5 * 60)
            wash_liquid.put("stop")
            collect_valve.put("position", params={"position": "Solvent"})
            await deliver_specific_vol(volume=3.2, last_full_withdraw=False,
                                       withdraw_p=r_valve_mapping["vial"],
                                       infuse_p=r_valve_mapping["waste"],
                                       withdraw_spd=1.0, infuse_spd=1.0,
                                       max_transfer_vol=1.0,
                                       wait_to_finish_infuse=True)

            collect_valve.put("position", params={"position": "Reagent"})
            wash_liquid.put("infuse", params={"rate": f"{5} ml/min"})
            await asyncio.sleep(3 / 5 * 60)
            wash_liquid.put("stop")
            collect_valve.put("position", params={"position": "Solvent"})
```

```
            await deliver_specific_vol(volume=3.2, last_full_withdraw=False,
                                       withdraw_p=r_valve_mapping["vial"],
                                       infuse_p=r_valve_mapping["analysis"],
                                       withdraw_spd=1.0, infuse_spd=1.0,
                                       max_transfer_vol=1.0,
                                       wait_to_finish_infuse=True)
```

```
In [ ]: async def system_log(date: datetime, mongo_id: str, total_time: float):
            import pandas as pd
            from requests import HTTPError
            end_time = time.monotonic() + total_time * 60
            inj_valve_mapping = {"load": 0, "inject": 1}
            ml600_valve_mapping = {"syr-left": 1, "syr-front": 2, "syr-right": 3}
            read_r2_sys = general_exp_sensor.get("monitor-system")

            if read_r2_sys.status_code == 500:
                default_data = {
                    "RunState_code": "0", "allValve": "00000",
                    "pumpA_P": 10, "pumpB_P": 10, "sysP (mbar)": 10, "Temp": 10,
                    "o2_flow": 0.0, "epc": 0.0, "air_flow": 0.0,
                    "pumpM_P": 0.0, "lcvalve": 0, "6portvalve": 0,
                }
                log = pd.DataFrame(default_data, index=[time.monotonic()])
            else:
                record = read_r2_sys.json()
                record["o2_flow"] = deliver_gas.get("get-flow-rate").json()
                record["epc"] = press_control.get("get-pressure").json()
                record["air_flow"] = press_helper.get("get-flow-rate").json()
                record["pumpM_P"] = pumpM_pressure.get(
                    "read-pressure").json()

                read_lcvalve = hplc_valve.get("position").json()
                record["lcvalve"] = inj_valve_mapping.get(read_lcvalve, 2)
                read_ml600 = transfer_valve.get("position").json()
                record["ml600left"] = ml600_valve_mapping.get(read_ml600, 0)

                log = pd.DataFrame(record, index=[time.monotonic()])
                await asyncio.sleep(1.0)

            while time.monotonic() < end_time:
                try:
                    read_sys = general_exp_sensor.get("monitor-system")
                    read_sys.raise_for_status()
                    n_record = read_sys.json()

                    n_record["o2_flow"] = deliver_gas.get("get-flow-rate").json()
                    n_record["epc"] = press_control.get("get-pressure").json()
                    n_record["air_flow"] = press_helper.get("get-flow-rate").json()
```

```
                    n_record["pumpM_P"] = pumpM_pressure.get("read-pressure").json()

                    read_lcvalve = hplc_valve.get("position").json()
                    n_record["lcvalve"] = inj_valve_mapping.get(read_lcvalve, 2)
                    read_ml600 = transfer_valve.get("position").json()
                    n_record["ml600left"] = ml600_valve_mapping.get(read_ml600, 0)

                    log = pd.concat([log, pd.DataFrame(n_record, index=[time.monotonic()])])
                    await asyncio.sleep(1.0)
                except HTTPError:
                    await asyncio.sleep(1.0)
                    continue
                finally:
                    log.to_csv(f'{date}_log_{mongo_id}.csv', header=True)

            logger.info("finish the log of system")
```

In [ ]:
```
async def main_exp():
    import Async_ClarityRemoteInterface
    hplc_commander = Async_ClarityRemoteInterface(
        remote=True, host='192.168.10.11',
        port=10015, instrument_number=1)

    exp_id = "whhsu_146_02_09"
    condition = {'concentration': 0.3, 'oxygen_equiv': 1.2,
                 'time': 1.0, 'wavelength': "440nm",
                 'light': 24, 'pressure': 2.5, 'temperature': 52,
                 }
    from BV_experiments.Example2_methionie.Example2_operating_para import Example2_cal
    # env_path = r"D:\BV\BV_experiments\Example2_methionie\pipeline_02.env"
    env_path = r"D:\BV\BV_experiments\Example2_methionie\pipeline.env"
    calculator = Example2_calculator(env_path, )

    # calculate the flow rate (g + l)
    rates = calculator.calc_gas_liquid_flow_rate(condition)
    schedule = calculator.reaction_schedule(condition, rates)

    total_rxn_time = schedule["pre_run_time"] + schedule[
        "fill_loop"] + schedule['loop_to_vial'] + schedule[
        "collect_time"]

    date = datetime.date.today().strftime("%Y%m%d")
    log_path = Path(f"{date}_{exp_id}.log")
    i = logger.add(log_path, rotation="10 MB")
    logger.info(f"exp.{exp_id}: {condition}")
    logger.info("_____________________________________________________")

    from BV_experiments.log_flow import flow_log
```

```
            await asyncio.gather(run_experiment(condition, calculator),
                                 system_log(date, exp_id, total_rxn_time),
                                 flow_log(date, exp_id, total_rxn_time)
                                 )
            await exp_hardware_init()
            logger.info(f"start prepare the hplc sample")

            await analyze_experiment(exp_id, condition, calculator, hplc_commander)

            # wash system
            await wash_system(3.0)
            await exp_hardware_init()
```

In [ ]:
```
if __name__ == "__main__":
    asyncio.run(main_exp())
```