Supporting Documentation

Method to Maintain Oxidative Capacity of Test Solutions for Oxidative Screening per Requirement of ISO 10993-13

S1. Python Code for Calibration Between H_2O_2 Concentration and Current Generated at the Electrodes

S1.1 This section presents the Python code to facilitate the calibration between H_2O_2 concentration and the current generated at the electrodes in response to the selected voltage protocol. It outlines the process of initializing the electrochemical system, setting calibration parameters, applying voltage protocols, and processing the resulting data to establish a calibration curve.

S1.2 Note that this code is written in Python 2.7. Given the discontinuation of support for Python 2.7 and potential compatibility issues with newer Python versions, users are encouraged to adapt the code to be compatible with their operational environment, preferably using a more recent version of Python, such as Python 3.x. This adaptation may involve syntax changes and updates to library calls to ensure the script's functionality in maintaining H_2O_2 concentration is preserved across different experimental setups and software configurations.

```
# Import necessary modules
from potentiostat import Potentiostat
import time
import serial
import serial.tools.list_ports
import traceback
potentiostat_ID = 0

# Electrochemical Run Parameters:
# These parameters define the settings for the potentiostatic experiment (electrochemical run) for each reaction module.
# Each module has a dictionary specifying the current range, voltage range, voltage limits, and corresponding time durations.
# - "curr_range": Current range for the potentiostatic experiment (e.g., '1000uA' for 1000 microamperes).
# - "volt_range": Voltage range for the potentiostatic experiment (e.g., '1V' for 1 volt).
# - "low_volt": Lower voltage limit during the experiment in volts.
# - "high_volt": Higher voltage limit during the experiment in volts.
# - "low_volt_time": Duration of the lower voltage limit in seconds.
# - "high_volt_time": Duration of the higher voltage limit in seconds.

curr_range = '1000uA'
volt_range = '1V'
low_voltage = -0.3
low_time = 0.5
high_voltage = 0.7
high_time = 2
sample_period = 0.1
```

```
# Define output file for results
filename = 'Add File Name here.txt'  # Adjusted filename

try:
    # Initialize potentiostat device
    dev = None
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
        print 'Checking port %s' % (str(p))
        try:
            dev_temp = Potentiostat(p[0])
            cT = dev_temp.get_device_id()
            if cT == potentiostat_ID:
                dev = dev_temp
            print 'Connected...'
        except:
            print 'Skipping...'
            pass

    if dev:
        # Configure potentiostat for chronoamperometry
        print 'Running chronoamperometry on potentiostat with device ID: %s' % (str(dev.get_device_id()))
        dev.set_all_elect_connected(True)
        dev.set_volt_range(volt_range)
        dev.set_curr_range(curr_range)

        # Define variables for data collection
        list_length = 5
        curr_list = None

        while True:
            stTime = time.time()
            run_pot = True
            hold_period = False
            sample_time = time.time()
            num_samples = 0
            total_curr = 0

            # Cycle for low_time + high_time
            dev.set_volt(low_voltage)
            while run_pot:
                if time.time() - stTime > low_time:
                    # Start holding high voltage
                    if not hold_period:
                        dev.set_volt(high_voltage)
                        hold_period = True
```

```
                    # Sample every sample_period seconds
                    if time.time() - stTime > low_time + (high_time/2):
                        if time.time() - sample_time > sample_period:
                            total_curr += dev.get_curr()
                            num_samples += 1
                            sample_time = time.time()

                # Exit cycle
                if time.time() - stTime > low_time + high_time:
                    run_pot = False

            # Calculate moving average of sampled currents
            if curr_list is None:
                curr_list = [total_curr/num_samples]
            elif len(curr_list) < list_length:
                curr_list = curr_list + [total_curr/num_samples]
            else:
                curr_list = curr_list[1:len(curr_list)] + [total_curr/num_samples]
            curr = sum(curr_list)/len(curr_list)

            # Write current value to the file
            with open(filename, 'a') as fp:
                fp.write("%f\n" % curr)
            print "%f" % (curr)
    else:
        print 'Could not connect potentiostat with device ID: %s' % (str(potentiostat_ID))
except Exception as e:
    # Print traceback if an exception occurs
    traceback.print_exc()
finally:
    # Ensure that the test is stopped on all available ports
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
        try:
            dev = Potentiostat(p[0])
            dev.stop_test()
        except:
            pass
```

S2. Python Code for Monitoring and Maintaining H_2O_2 Concentration

S2.1 This section presents the Python code used for maintaining H_2O_2 concentration within specified electrochemical experiments. The code exemplifies the integration of software control with the experimental setup to regulate H_2O_2 levels dynamically. It encompasses initializing the system, setting experimental parameters based on prior calibration, conducting continuous electrochemical monitoring, and activating adjustment mechanisms to maintain the target concentration.

S2.2 Note that this code is written in Python 2.7. Given the discontinuation of support for Python 2.7 and potential compatibility issues with newer Python versions, users are encouraged to adapt the code to be compatible with their operational environment, preferably using a more recent version of Python, such as Python 3.x. This adaptation may involve syntax changes and updates to library calls to ensure the script's functionality in maintaining H_2O_2 concentration is preserved across different experimental setups and software configurations.

```
# Import necessary modules
import RPi.GPIO as GPIO
import time
from datetime import datetime
import smbus
import serial
import serial.tools.list_ports
import traceback
from potentiostat import Potentiostat

# Vessel Parameters
experimentName = ['Add file name here']
experimentRunParameters = [[145]]  # Set target H2O2 concentration here
average_value = 20 # This calculates the running average of measured H2O2 concentration

# Calibration parameters for converting current to H2O2 concentration
# The values below (12.292 and 0.1999) are examples and should be edited based on the user's calibration experiment.
# These values represent the y-intercept and slope of the linear regression between current and H2O2 concentration.
# Calibration Experiment: y = mx + b, where y is H2O2 concentration, x is current.
experimentCurrToConcFunctions = [lambda x: (x - 12.292) / 0.1999]

# Electrochemical Run Parameters:
# These parameters define the settings for the potentiostatic experiment (electrochemical run) for each reaction module.
# Each module has a dictionary specifying the current range, voltage range, voltage limits, and corresponding time durations.
# - "curr_range": Current range for the potentiostatic experiment (e.g., '1000uA' for 1000 microamperes).
# - "volt_range": Voltage range for the potentiostatic experiment (e.g., '1V' for 1 volt).
# - "low_volt": Lower voltage limit during the experiment in volts.
```

```
# - "high_volt": Higher voltage limit during the experiment in volts.
# - "low_volt_time": Duration of the lower voltage limit in seconds.
# - "high_volt_time": Duration of the higher voltage limit in seconds.
echemRunParameters = [{"curr_range": '1000uA',
                       "volt_range": '1V',
                       "low_volt": -0.3,
                       "high_volt": 0.7,
                       "low_volt_time": .5,
                       "high_volt_time": 2}]

# GPIO pins corresponding to peristaltic pump control
# Pin 17 is an example and should be edited based on the user's setup.
# This pin has positive voltage and is connected to an external power relay
# that controls the peristaltic pump.
pumpPins = [17]

defaultPath = '/home/pi/potentiostat-master/software/python/potentiostat/RAA'

# Sampling rate for collecting data from the potentiostat. This is the rate
# at which the system reads potential and current data from the potentiostat.
# For example, the code below specifies the sampling rate (sampleRate) as float(2.5**-1),
# which is equivalent to 1/2.5 or approximately 0.4 Hz.
# This means that the system collects data from the potentiostat every 1/0.4 seconds,
# or approximately every 2.5 seconds.
sampleRate = float(2.5**-1)

# Data write rate for storing collected data to a file. This is the rate
# at which the system writes the collected data to a file.
# For example, the code below specifies the sampling rate (dataWriteRate) as float(15**-1),
# which is equivalent to 1/15 or approximately 0.067 Hz.
# This means that the system writes data to the file every 1/0.067 seconds,
# or approximately every 15 seconds.
dataWriteRate = float(15**-1)

# I2C bus setup
bus = smbus.SMBus(1)
numberOfRAAs = len(experimentName)
allH2O2Conc = [[] for _ in range(numberOfRAAs)]
all_current = [float('nan') for _ in range(numberOfRAAs)]
echem_running_status = [False for _ in range(numberOfRAAs)]
echem_timing = [[] for _ in range(numberOfRAAs)]
new_current_available = [False for _ in range(numberOfRAAs)]
estimated_echem_end_time = [[] for _ in range(numberOfRAAs)]

# File names for data logging
dataLogFileName = [defaultPath + '/' + exp_name + '.txt' for exp_name in experimentName]
GPIO.setmode(GPIO.BCM)
```

```
# Set up GPIO pins for pumps
for k in range(len(pumpPins)):
    GPIO.setup(pumpPins[k], GPIO.OUT, initial=GPIO.LOW)

# Function to find Serial ports based on PID
def find_PID_serial_ports(num_RAAs):
    ser_ports = list(serial.tools.list_ports.comports())
    serialPID = [[] for _ in range(num_RAAs)]
    for k in range(len(ser_ports)):
        for k1 in range(num_RAAs):
            ser = serial.Serial(ser_ports[k][0], 9600, timeout=0)
            ser.flushInput()
            ser.flushOutput()
            cT = ser.readlines()
            string = f'*00{k1 + 1}G110 \r\r'
            ser.flushInput()
            ser.flushOutput()
            ser.write(string)
            ser.flushInput()
            ser.flushOutput()
            time.sleep(.1)
            cT = ser.readlines()
            ser.flushInput()
            ser.flushOutput()
            ser.close()
            if cT:
                serialPID[k1] = ser_ports[k][0]
    return serialPID

# Function to find Arduino Serial ports
def find_arduino_serial_ports(num_RAAs):
    Ard_Address = [[] for _ in range(num_RAAs)]
    ports = list(serial.tools.list_ports.comports())
    print('Looking for ARD port...')
    for p in ports:
        try:
            dev = Potentiostat(p[0])
            cT = dev.get_device_id()
            if float(cT) == 0:
                print(f'Connecting RAA 1 to Arduino (ID: {cT}) via port {p[0]}.')
                Ard_Address[0] = dev
        except:
            pass
    for i in range(num_RAAs):
        if not Ard_Address[i]:
            print(f'Could not connect RAA {i + 1} to potentiostat')
    return Ard_Address
```

```
# Function to write data to RAA file
def Write_To_RAA_File(log_file_name, new_line):
    current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]
    new_line = [current_time] + [str(val) for val in new_line]
    new_line = '\t'.join(new_line) + '\n'
    with open(log_file_name, 'a') as RAAFile:
        RAAFile.write(new_line)

# Function to add value to data list
def Add_Value_To_Data_List(old_vals, new_val, len_limit):
    if len(old_vals) >= len_limit:
        old_vals = old_vals[1:]
    old_vals.append(new_val)
    return old_vals

# Function to run experiment in synchronized mode
def run_rodeo_synchronized(device_ID, echem_timer, echem_parameters):
    # ... (skipped for brevity)

# Function to initialize the experiment
def initialize_experiment(device_ID, pump_pin, experiment_parameters):
    # ... (skipped for brevity)

# Find Serial ports for PID and Arduino
serialPID = find_PID_serial_ports(1)  # Changed to 1 RAA
Ard_Address = find_arduino_serial_ports(1)  # Changed to 1 RAA

# Check if connections are successful
if not serialPID[0]:
    print('Could not connect to RAA 1')
    GPIO.cleanup()
    quit()

# Initialize experiment for the single RAA
pumpPin = pumpPins[0]
deviceID = Ard_Address[0]
experimentRunParameter = experimentRunParameters[0]
experimentCurrToConcFunction = experimentCurrToConcFunctions[0]
echemRunParameter = echemRunParameters[0]
experimentName_k = experimentName[0]
dataLogFileName_k = dataLogFileName[0]

experiment_parameters = {
    'curr_to_conc_function': experimentCurrToConcFunction,
    'echem_parameters': echemRunParameter,
    'experiment_run_parameters': experimentRunParameter,
    'experiment_name': experimentName_k,
    'data_log_file_name': dataLogFileName_k,
```

```
    'sample_rate': sampleRate,
    'data_write_rate': dataWriteRate
}

try:
    device_ID = Potentiostat(serialPID[0])
    initialize_experiment(device_ID, pumpPin, experiment_parameters)
except:
    print(f"Error in RAA 1: {traceback.format_exc()}")
    pass

finally:
    GPIO.cleanup()
    print('Exiting script...')
```

S3. Process Flow for the Control Script for Calibration Between H_2O_2 Concentration and Current Generated at the Electrodes

S3.1 This section outlines the process flow for the control script used in the electrochemical calibration of hydrogen peroxide (H_2O_2) concentration. The flowchart provided herein details the sequential steps executed by the script, from initializing the electrochemical system to the final data analysis and verification stages. It serves as a guide for replicating the calibration procedure within the specified test method and is intended to assist users in understanding the software-driven operations that underpin the electrochemical measurements. Each step in the flowchart corresponds to a critical function of the control script, ensuring that the system operates accurately and efficiently to maintain the integrity of the calibration process. The script's primary functions include:

- *S3.1.1 Initialization and Connection*: Initiates communication with and connects to the electrochemical device, identifying the device using its unique identifier.
- *S3.1.2 Parameter Configuration:* Defines the parameters for the electrochemical experiment, including voltage protocol and timing for data collection.
- *S3.1.3 Voltage Application and Data Collection:* Applies the predetermined voltage protocol, while simultaneously recording the current response.
- *S3.1.4 Data Processing and Analysis:* Processes the recorded data to calculate current.
- *S3.1.5 Decision Point:* The script reaches a critical decision point where it assesses whether additional data are required. This determination is based on user-defined criteria, which could include the total number of samples collected, the consistency of the data, or the attainment of a certain level of statistical significance.
- *S3.1.6 Experiment Conclusion:* If the script determines that no further data collection is necessary, it proceeds to safely conclude the experiment, ensuring the system's operation is properly terminated.

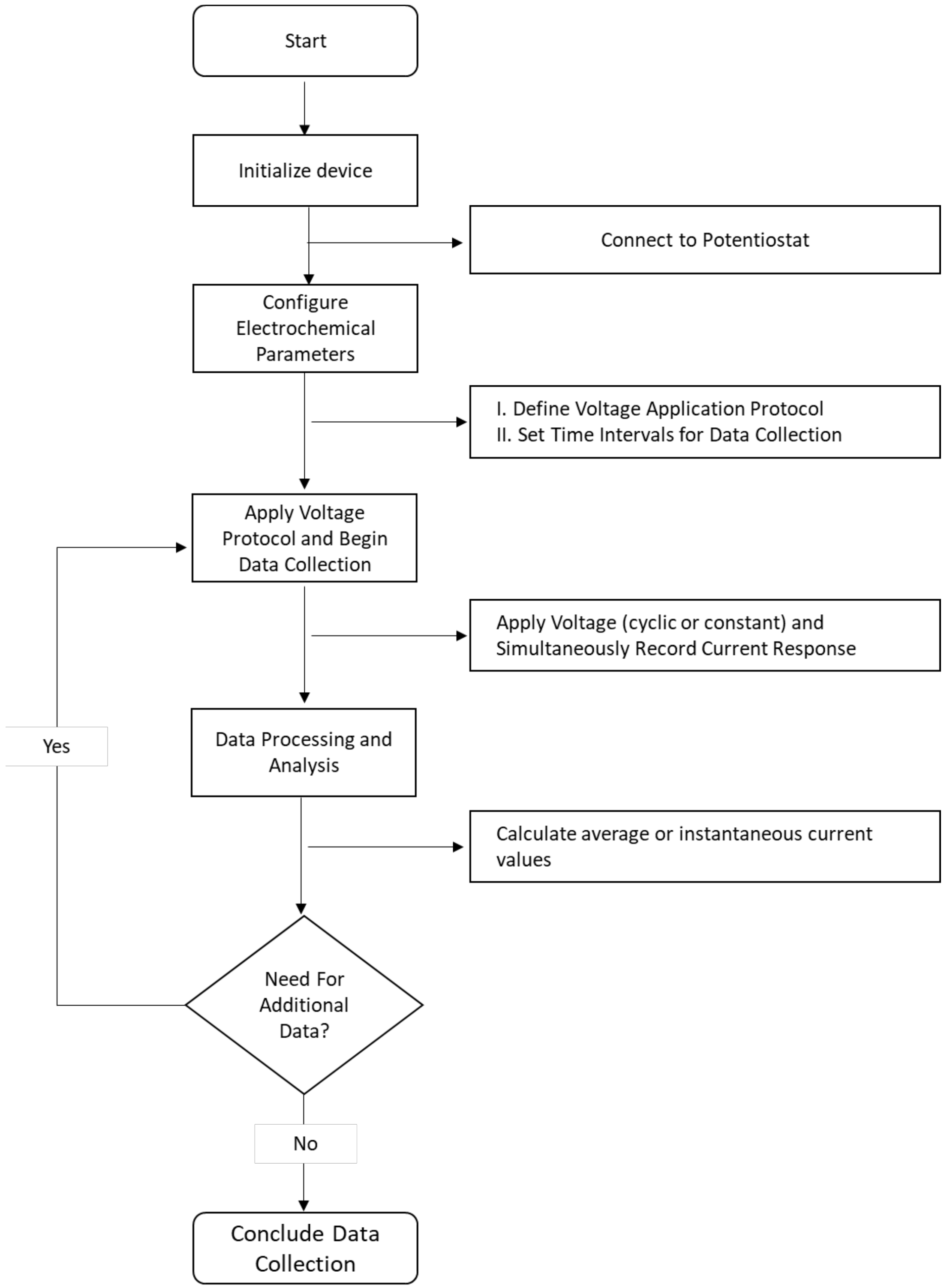

Figure S1 – Flowchart for the control script for calibration between H_2O_2 concentration and current generated at the electrodes.

S4. Process Flow for the Control Script for Maintaining and Monitoring H_2O_2 Concentration

S4.1 This appendix outlines the process flow for the control script that is responsible for maintaining the concentration of H_2O_2 during electrochemical experiments. This script integrates with the hardware setup to regulate the H_2O_2 concentration by managing the peristaltic pump operations based on real-time data analysis. The flowchart illustrates the sequence of operations from the initialization of the control system, through the application of electrochemical parameters, to the activation of corrective measures when H_2O_2 levels deviate from the target range. Key elements of the control script include:

- *S4.1.1 Initialization and Connection*: Initiates communication with and connects to the electrochemical device, identifying the device using its unique identifier. This step also involves configuring the activation of the pump by the computing system.
- *S4.1.2 Parameter Configuration:* Inputs calibration constants and experimental parameters such as target H_2O_2 concentration and voltage application protocols.
- *S4.1.3 Voltage Application and Data Collection:* Applies the predetermined voltage protocol, while simultaneously recording the current response.
- *S4.1.4 Data Analysis and Pump Activation:* Processes the recorded data to calculate H_2O_2 concentration and activates the peristaltic pump if the concentration is lower than the target H_2O_2 concentration.
- *S4.1.5 Monitoring and Adjustment:* Continuously monitors the system, executing the control loop that includes data collection, analysis, and pump activation as needed, based on the real-time data.
- *S4.1.6 Experiment Conclusion:* Concludes data collection and pump operation based on user-defined criteria tailored to the experimental objectives, such as a specified duration or achieving a steady-state concentration.

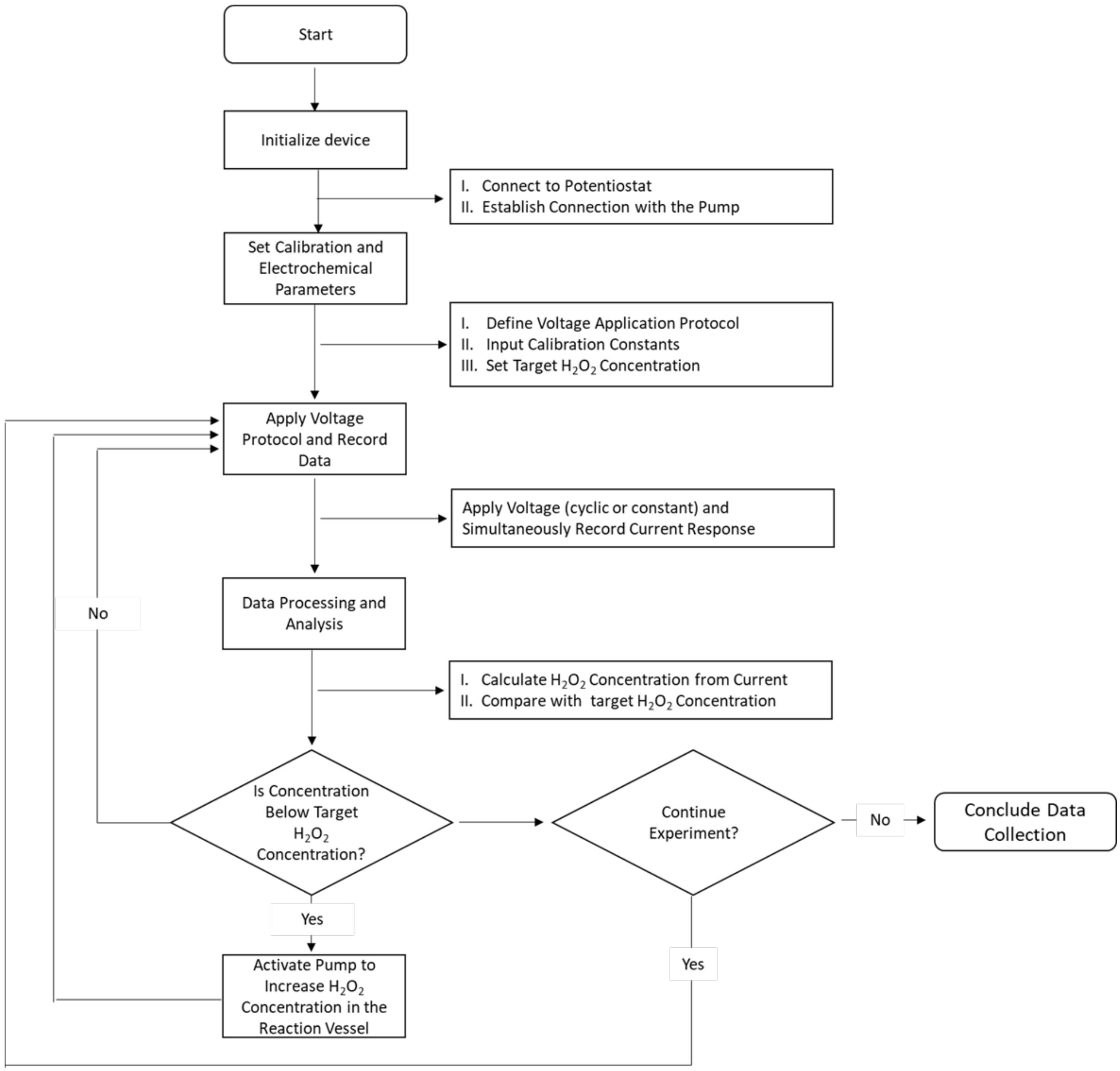

Figure S2 – Flowchart for the control script for maintaining and monitoring H_2O_2 concentration.