SASTA OLIPHANT
SCIENCE AWARDS
SOUTH AUSTRALIAN SCIENCE TEACHERS ASSOCIATION

Prize Winner

Programming, Apps & Robotics

Year 9-10

Ginger Vallance

Heathfield High School

ROWE SCIENTIFIC PTY. LTD. www.rowe.com.au

Australian Government
Department of Defence

Government of South Australia
Department for Education

SASTA South Australian Science Teachers Association

Oliphant Photobioreactors

Aim

The purpose of this build is to create a monitoring system for a photobioreactor. A photobioreactor is a bioreactor with the purpose of sustaining different kinds of algae[3]. The reason this is being monitored is to keep the algae at optimum conditions, in the case of chaetoceros caulcitranaes a food and biofuel source, 25'C[2] is optimal. This is achieved by a series of sensors monitoring the algae and alerting the user if there is excess or deficiency of any one the monitored variables. Due to constraints on time and budget, a temperature sensor will be used as a demonstration, in further iterations sensors such as pH and dissolved oxygen.

This could be applied to many industries which use general of microalgae for example agriculture[1], biofuels, medicine and wastewater treatment[4]. A similar system could be used to monitor other forms of bioreactors such as ones containing bacteria which could be used further for medical research.

There are many scientific applications for a monitored photobioreactor, one of the main is changing variables within the algae. This could be used to stress the algae to increase lipid content, which is useful for biofuels. It could also be used to see how long it takes for algae to recover from stressful environments, this could be used to study ecosystems, such as swamps, that use algae as a producer. It could additionally be used for monitoring the increase of algae growth under different conditions to find an optimal growth rate for industries such as agricultural.

Specifications

- Raspberry pi 3B with raspberry pi OS installed
- DS18B20
- Bread Board

Instructions

To load the program open the file in Thonny and run it, due to Thonny picking up a non-existent error in redundancy, the user must click onto the Thonny application before returning to the window.

Next the user can interact with the buttons to get the desired effect.

- The entry box with the label 'Interval (s)' next to it allows the user to change the amount of seconds between the taking and graphing of data points, the automatic setting is 1. If the user enters an invalid number, such as a negative, they will receive a message saying as such.
- The entry box with the label 'Max Data Points' next to it allows the user to set a number of data points they would like to graph. It is not required to have this field filled and can be stopped at any time with a separate button. The user must enter a positive integer or the program will display a message telling them to. Once collection is complete the user will be informed with a popup.
- The button labelled 'Start collection' begins to take data from the sensor files and places them onto the graph every three seconds when clicked.

- The button labelled ‘Stop collection’ stops collecting and graphing data, as well as notifying the user this has happened when clicked.
- The button ‘Reset Graph’ clears all of the temperature data from the graph when clicked.
- The button ‘Save Graph’ allows the user to save a png of the graph in a ‘save file’ popup
- The button ‘View Logs’ allows the user to see all of the data logs from the current day, the rest are stored in a file.
- The button ‘Quit’ exits the user from the program.

Code

A list of imports including tkinter for the GUI, matplotlib for the graphing, time/datetime to put onto the x axis of the graph when collecting data and the Operating system (os) and glob to access where the sensor is stored.

```
import tkinter as tk

from tkinter import messagebox, filedialog, scrolledtext

import os

import glob

import time

import threading

import datetime as dt

import matplotlib

matplotlib.use("TkAgg")

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

This access the operating system and activates the 1-wire interface for the sensor.

```
os.system('modprobe w1-gpio')

os.system('modprobe w1-therm')
```

This looks within the operating system files for the DS18B20 device, they start with 28.

```
base_dir = '/sys/bus/w1/devices/'

device_folders = glob.glob(base_dir + '28*')
```

If the sensor is not found it uses ‘dummy’ temperature readings in order to test the device.

```
if not device_folders:

    print("No DS18B20 sensor found. Using dummy temperature readings.")
```

```
  def read_temp(): return 25.0
  device_file = None
```

If the sensor is found it defines the folder to store the raw data from the sensor

```
else:
  device_folder = device_folders[0]
  device_file = device_folder + '/w1_slave'
```

This reads the data from the sensor's data file and checks for 'yes' to confirm that the data is valid before extracting it and returning it in 'C.

```
  def read_temp_raw():
    with open(device_file, 'r') as f:
      lines = f.readlines()
    return lines


  def read_temp():
    lines = read_temp_raw()
    attempts = 0
    while lines[0].strip()[-3:] != 'YES':
      time.sleep(0.2)
      lines = read_temp_raw()
      attempts += 1
      if attempts > 10:
        raise IOError("Failed to read sensor data after multiple attempts.")


    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
      temp_string = lines[1][equals_pos + 2:]
      try:
        temp_c = float(temp_string) / 1000.0
        return temp_c
      except ValueError:
        raise ValueError(f"Could not parse temperature from '{temp_string}'")
    else:
      raise ValueError("Could not find 't=' in sensor output.")
```

This lists and stores the timestamps and temperature values in the x and y values. It then sees whether data is being collected. The code then sets the number of seconds between each data point being taken and can optionally set a limit on the number of data points being taken. Finally it prevents data and plot updates from clashes across threads, as this was an issue previously.

```
xs = []
ys = []
collecting = False
interval = 1
max_points = None
lock = threading.Lock()
plot_lock = threading.Lock()
```

This initializes the tkinter window and sets the size and title of the GUI, as well as the font.

```
root = tk.Tk()
root.title("Live Temperature Logger")
root.geometry("800x400")

font_style = ("Arial", 12)

tk.Label(root, text="Interval (s):", font=font_style).grid(row=0, column=0, sticky='e', padx=5, pady=5)
```

These are the entry fields for the user to edit the intervals between collected data points and the amount of data points taken.

```
entry_interval = tk.Entry(root, font=font_style, width=10)
entry_interval.grid(row=0, column=1, padx=5, pady=5)
entry_interval.insert(0, "1")

tk.Label(root, text="Max Data Points:", font=font_style).grid(row=0, column=2, sticky='e', padx=5,
pady=5)
```

```
entry_max_points = tk.Entry(root, font=font_style, width=10)

entry_max_points.grid(row=0, column=3, padx=5, pady=5)

entry_max_points.insert(0, "")
```

This sets the graph up to display within the GUI, the line is updated with new temperature values and the canvas embeds the plot into the previous window. The line “marker='o', linestyle='-')” causes Thonny to read an error message in shell saying that this is redundant. The code would not function the same without the line, meaning that when the program is started the user must switch tabs back to Thonny before they can interact with the window.

```
fig, ax = plt.subplots(figsize=(7.8, 2.3), dpi=100)

line, = ax.plot_date([], [], marker='o', linestyle='-')

canvas = FigureCanvasTkAgg(fig, master=root)

canvas_widget = canvas.get_tk_widget()

canvas_widget.grid(row=3, column=0, columnspan=6, sticky='nsew', padx=5, pady=5)
```

This ensures that the graph area resizes properly with the window, preventing the data points from going off the side of the screen.

```
root.grid_rowconfigure(3, weight=1)

root.grid_columnconfigure(0, weight=1)

root.grid_columnconfigure(1, weight=1)

root.grid_columnconfigure(2, weight=1)

root.grid_columnconfigure(3, weight=1)
```

This sets the initial labels title and formatting, as well as ensuring consistent layout and axis limits. In this particular instance the graph title has been set to ‘Temperature in Bioreactor’ as this is what the device was tested with, however, this can be changed to suit the use, such as ‘Temperature in FishTank’.

```
def setup_initial_plot_state():

  ax.set_title("Temperature in Bioreactor")

  ax.set_ylabel("Temperature (°C)")

  ax.set_xlabel("Time")

  ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))

  ax.tick_params(axis='x', rotation=45)
```

```
    now = dt.datetime.now()
    ax.set_xlim(now - dt.timedelta(minutes=1), now + dt.timedelta(seconds=10))
    ax.set_ylim(20, 30)
    fig.tight_layout()

setup_initial_plot_state()
```

This updates the code every 3 seconds the graph is updated, this is to prevent the graph from freezing with the amount of data it is taking in. It also uses thread safe copies of xs and ys for this reason.

```
def update_graph():
    with plot_lock:
        with lock:
            current_xs = list(xs)
            current_ys = list(ys)

        if current_xs and current_ys:
            line.set_data(current_xs, current_ys)
            ax.relim()
            ax.autoscale_view(scalex=False, scaley=True)

            if len(current_xs) > 1:
                end_time = current_xs[-1] + dt.timedelta(seconds=5)
                start_time = end_time - dt.timedelta(seconds=65)
                ax.set_xlim(start_time, end_time)
            elif current_xs:
                ax.set_xlim(current_xs[0] - dt.timedelta(seconds=30), current_xs[0] +
dt.timedelta(seconds=30))
            else:
                setup_initial_plot_state()

            ax.set_title("Temperature in Bioreactor")
            ax.set_ylabel("Temperature (°C)")
            ax.set_xlabel("Time")
            ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))
            ax.tick_params(axis='x', rotation=45)
        else:
            line.set_data([], [])
            setup_initial_plot_state()

        fig.tight_layout()
        canvas.draw()
    root.after(3000, update_graph)
```

This appends the timestamp and temperature to a .csv file labelled by the date and writes a header if the file is new. This is so the user can go through previous logs if they are looking for changes in the data.

```
def log_to_file(timestamp, temp):
    log_filename = f"temperature_log_{dt.datetime.now().date()}.csv"
    write_header = not os.path.exists(log_filename)

    try:
        with open(log_filename, 'a') as f:
            if write_header:
                f.write("Timestamp,Temperature(C)\n")
            f.write(f"{timestamp.strftime('%H:%M:%S')},{temp:.2f}\n")
    except IOError as e:
        print(f"Error writing to log file {log_filename}: {e}")
```

This is a background loop that reads the temperature appends it to xs and ys logs it to the file and stops when the max data points is reached, as well as sleeping for the specified interval.

```
def collect_data():
    global collecting
    point_count = 0
    while collecting:
        try:
            temp = read_temp()
            timestamp = dt.datetime.now()
            with lock:
                xs.append(timestamp)
                ys.append(temp)
                if len(xs) > 60:
                    xs.pop(0)
                    ys.pop(0)
            log_to_file(timestamp, temp)
            print(f"{timestamp.strftime('%H:%M:%S')}: {temp:.2f} °C")
            point_count += 1
            if max_points is not None and point_count >= max_points:
                collecting = False
                root.after(0, lambda: messagebox.showinfo("Collection Finished", f"Collected {max_points} data points."))
                break
        except (IOError, ValueError) as e:
            print(f"Error reading sensor: {e}")
        time.sleep(interval)
```

This validates the user input for the interval and max points and starts 'collect_data()' in a thread, meaning that it is collecting the data for the amount of time and within the intervals specified by the user. It also tells the user that the interval and max data points must be a positive number and integer respectively if they enter an invalid number, such as -9.

```
def start_collection_thread():
    global collecting, interval, max_points
    try:
        interval = float(entry_interval.get())
        if interval <= 0:
            raise ValueError("Interval must be a positive number.")
        max_input_str = entry_max_points.get().strip()
        if max_input_str:
            try:
                max_points_candidate = int(max_input_str)
                if max_points_candidate <= 0:
                    raise ValueError("Max Data Points must be a positive integer.")
                max_points = max_points_candidate
            except ValueError:
                raise ValueError("Max Data Points must be a whole number (integer).")
        else:
            max_points = None

    except ValueError as e:
        messagebox.showerror("Invalid input", f"Input error: {e}")
        return

    if not collecting:
        collecting = True
        threading.Thread(target=collect_data, daemon=True).start()
```

This stops the collection sequence and notifies the user via a pop up.

```
def stop_collection():
    global collecting
    if collecting:
        collecting = False
        messagebox.showinfo("Collection Stopped", "Data collection has been stopped.")
```

This opens a file saving dialog and allows the user to save the current data and saves as a png with a name and file of their choice.

```
def save_plot_on_main_thread():
    if not xs:
        messagebox.showwarning("No data", "No data to save.")
        return

    filename = filedialog.asksaveasfilename(
        defaultextension=".png",
        filetypes=[("PNG files", "*.png")],
        title="Save Graph As"
    )

    if filename:
        try:
            with plot_lock:
                with lock:
                    line.set_data(xs, ys)
                ax.relim()
                ax.autoscale_view()
                ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M:%S'))
                ax.tick_params(axis='x', rotation=45)
                fig.tight_layout()
                canvas.draw()

                fig.savefig(filename)

            root.update_idletasks()
            messagebox.showinfo("Saved", f"Graph saved as {filename}")

        except Exception as e:
            messagebox.showerror("Error", f"Failed to save graph: {e}")

def threaded_save_plot():
    root.after(0, save_plot_on_main_thread)
```

This opens the saved logs from the day in a new window and uses a thread to keep the GUI responsive as previously it would stop scrolling.

```
def threaded_view_logs():
    threading.Thread(target=view_logs, daemon=True).start()


def view_logs():
    log_filename = f"temperature_log_{dt.datetime.now().date()}.csv"
    if not os.path.exists(log_filename):
        messagebox.showinfo("No log", "No log file found for today.")
        return

    root.after(0, lambda: _open_and_display_log(log_filename))


def _open_and_display_log(log_filename):
    log_window = tk.Toplevel(root)
    log_window.title("View Logs")
    log_window.geometry("500x300")

    text_area = scrolledtext.ScrolledText(log_window, wrap=tk.WORD, font=("Courier", 10))
    text_area.pack(expand=True, fill='both')

    try:
        with open(log_filename, 'r') as f:
            contents = f.read()
            text_area.insert(tk.END, contents)
            text_area.see(tk.END)
    except Exception as e:
        messagebox.showerror("Error", f"Failed to read log file: {e}")
        log_window.destroy()
```

This stops collection, clears the data and sets the plot to default.

```
def reset_graph():
    global xs, ys, collecting
    if collecting:
        stop_collection()

    with lock:
        xs.clear()
        ys.clear()

    with plot_lock:
        line.set_data([], [])
        setup_initial_plot_state()
        canvas.draw()

    messagebox.showinfo("Graph Reset", "The graph has been reset.")
```

This allows the user to stop collection and exit the application

```
def quit_application():
    global collecting
    if collecting:
        stop_collection()
    root.quit()
    root.destroy()
```

This the buttons and what should happen if triggered.

```
tk.Button(root, text="Start Collection", font=font_style,
command=start_collection_thread).grid(row=1, column=0, pady=10, padx=5)

tk.Button(root, text="Stop Collection", font=font_style, command=stop_collection).grid(row=1,
column=1, pady=10, padx=5)
```

```
tk.Button(root, text="Save Graph", font=font_style, command=threaded_save_plot).grid(row=1, column=2, pady=10, padx=5)

tk.Button(root, text="View Logs", font=font_style, command=threaded_view_logs).grid(row=1, column=3, pady=10, padx=5)


tk.Button(root, text="Reset Graph", font=font_style, command=reset_graph).grid(row=1, column=4, pady=10, padx=5)

tk.Button(root, text="Quit", font=font_style, command=quit_application).grid(row=1, column=5, pady=10, padx=5)
```

This begins to update the graph and GUI in a loop.

```
root.after(1000, update_graph)

root.mainloop()
```

Acknowledgements

Acknowledgment of any external support provided to the entry

Scott Vallance, Funding

Francine Vallance, Funding

Bibliography

A bibliography that acknowledges relevant sources of information

1. de Oliveira, A.P.F. and Bragotto, A.P.A. (2022). MICROALGAE-BASED PRODUCTS: FOOD AND PUBLIC HEALTH. *Future Foods*, p.100157. doi:https://doi.org/10.1016/j.fufo.2022.100157.

2. Krichnavaruk, S., Loataweesup, W., Powtongsook, S. and Pavasant, P. (2005). Optimal growth conditions and the cultivation of Chaetoceros calcitrans in airlift photobioreactor. *Chemical Engineering Journal*, 105(3), pp.91–98. doi:https://doi.org/10.1016/j.cej.2004.10.002.

3. Chanquia, SN, Vernet, G & Kara, S 2021, ‘Photobioreactors for cultivation and synthesis: Specifications, challenges, and perspectives’, *Engineering in Life Sciences*, vol. 22, Wiley, no. 12, pp. 712–724, viewed 29 May 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC9731602/#:~:text=Photobioreactors%20are%20defined%2C%20in%20most,lights%20are%20a%20better%20choice.>.

4. Wang, X, Ma, S & Kong, F 2024, ‘Microalgae Biotechnology: Methods and Applications’, *Bioengineering*, vol. 11, MDPI AG, no. 10, p. 965, viewed 29 May 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC11506088/#:~:text=This%20superior%20bacterial%20inhibition%20efficacy,applications%20and%20sustainable%20production%20methods.>.