Microfluidic platform for screening the activity of immobilized photocatalysts for degradation of water pollutants in flow

Anca Roibu,*[a] Razvan Udroiu [b], Alexandru Dinu [c] and Luminita Andronic *[a]

a. Product Design, Mechatronics and Environment Department, Transilvania University of Brasov, Romania.
b. Manufacturing Engineering Department, Transilvania University of Brasov, Romania.
c. Electronics and Computers Department, Transilvania University of Brasov, Romania.
* Corresponding authors: Anca Roibu (e-mail: anca.roibu@unitbv.ro), Luminita Andronic (e-mail: andronic-luminita@unitbv.ro).

Outline

S1. Experimental setup

The microfluidic platform components and how are they connected can be visualized in Figure S1. The syringe pump can be connected with each of the four microreactors, M1, M2, M3, and M4. In this study, the microreactors are operated one at a time. A simultaneous pumping into the four microreactors was attempted using a 5-Port Manifold Body for 1/16 inch OD Tubing (Upchurch Scientific). However, the flow would be similar only in 2-3 microreactors and much lower in the fourth one. By consecutively using four solenoid valves we improved the flow distribution in the four microreactors, but a difference in the flow rates was still observed along with different photodegradation efficiencies. For achieving the operation of the four microreactors using a single syringe pump, additional efforts will be invested in the future in designing a suitable flow distribution.

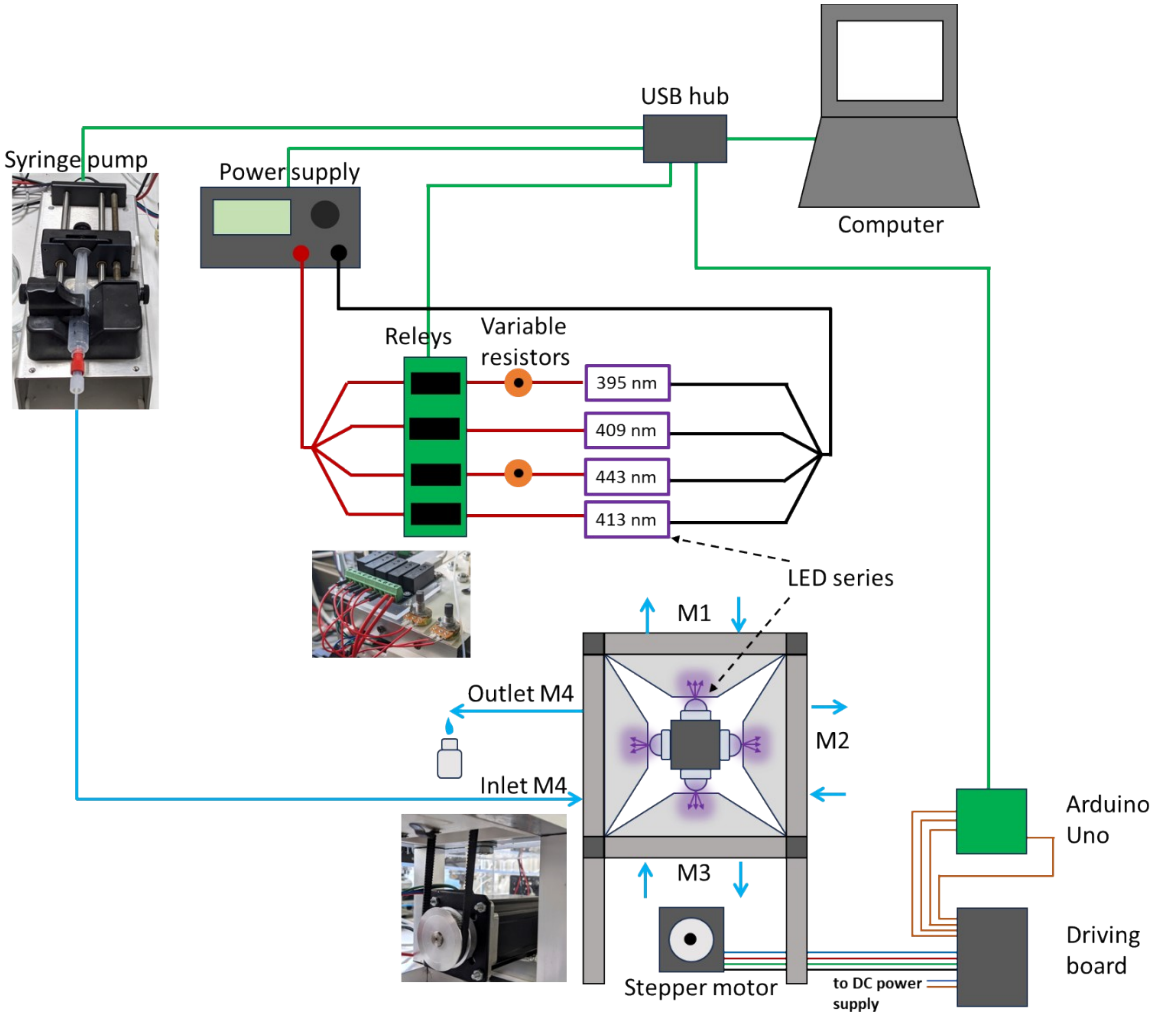

Figure S1. Schematic representation of the experimental setup.

Figure S2. Image of the ansambled 3D-printed microreactor that includes the TiO_2 photocatalyst immobilized on a glass plate.

When the microreactor is assembled using the 3D-printed components and TiO_2 photocatalyst immobilized on the glass plate, a window is formed where the photocatalyst is exposed to the LEDs radiation as shown in Figure S2.

Two 3D-printed covers were added on the sides of the screening platform to minimize the effect of the external light on the experiments (Figure S3). Its manufacturing time was 5h 56 min and the infill density was 20%. The other manufacturing parameters were similar to the microreactor parts. The 3D model of the covers is illustrated in Figure S3a. On one cover, four radial fans were attached to cool down by air the four sides of the LED light source (Figure S3b). On the other cover, several openings were created to enable air circulation (Figure S3c).

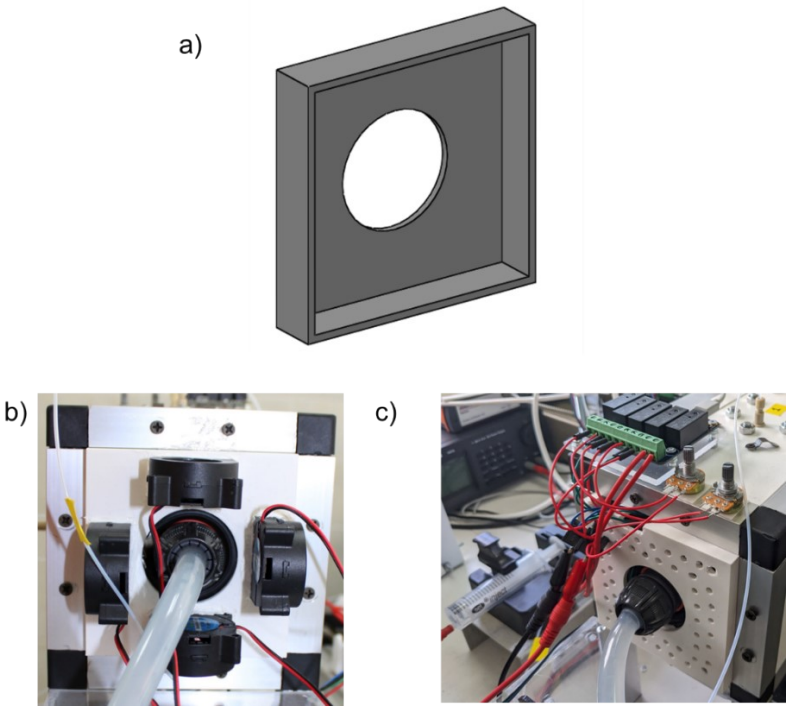

Figure S3. The 3D printed side covers of the microfluidic platform. a) The 3D model of the side cover, b) image of the side cover attached to radial fans, and c) image of the side cover containing openings for air circulation.

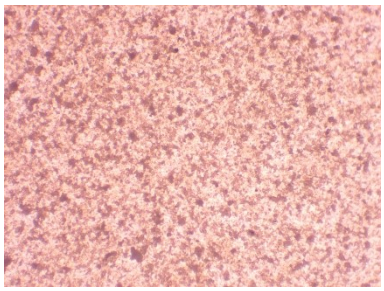

Figure S4. Microscope image of the immobilized TiO_2 P25/20 film.

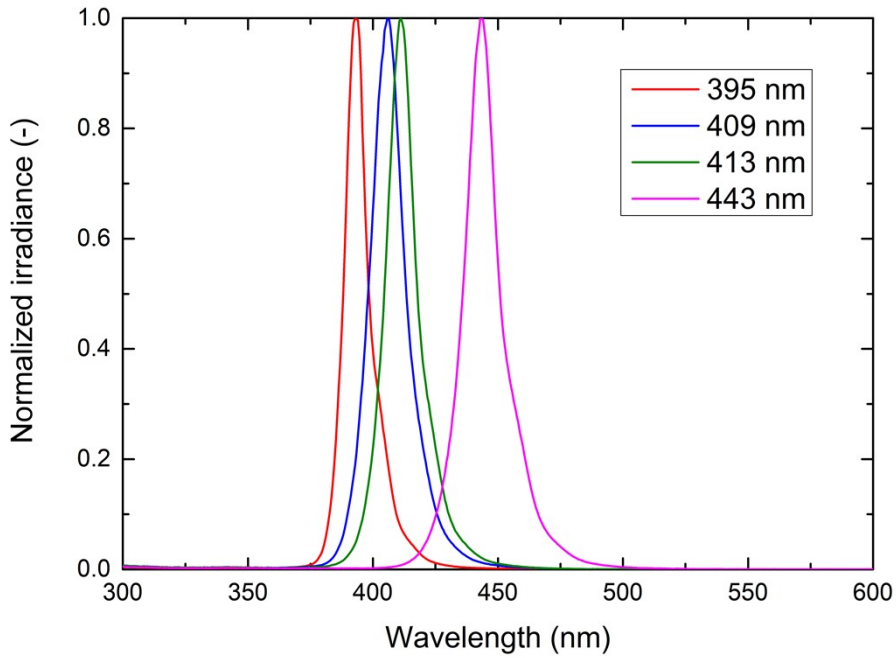

Figure S5. Spectral distribution of the LED light source emitting at 395, 409, 413, and 443 nm.

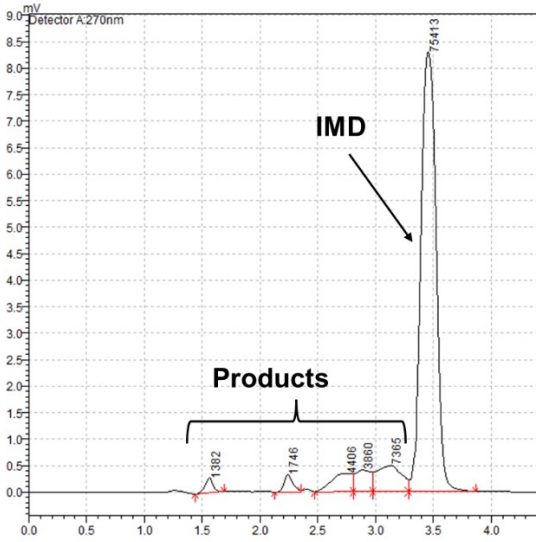

Figure S6. An example of a HPLC chromatogram for the solution resulted from a photocatalytic test.

The IMD degradation efficiency was calculated based on the reduction of the IMD peak area. The apparition of several products was observed in the chromatogram as well (see Figure S6); however, the products were not identified due to the complex degradation mechanism of IMD[1].

S2. Controlling the screening platform components

To control the components of the screening platform such as the syringe pump, power supply for LEDs, relays, and stepper motor (via an Arduino Uno board), they were connected to a computer via an USB hub as shown in Figure S1 (see green lines).

To start, stop, and set the operating parameters of the **syringe pump**, we used WinPumpTerm software or Python. We used and adapted the functions provided at https://github.com/RomeroLab/syringe-pump-controller/blob/master/new_era.py. Several functions for controlling the syringe pump using Python are listed below:

```
def set_diameter(ser,pump,dia):
    cmd = b'%iDIA%s\x0D'%(pump,dia)
    ser.write(cmd)
    output = ser.readline()
    if b'?' in output: print (str(cmd.strip())+' from set_diameter not
understood')
    else:
        time.sleep(1)
        print("Diameter of pump was set to be:", get_diameter(ser,pump));

def set_direction(ser,pump, direction):
    frcmd = b'%iDIR%s\x0D'%(pump,direction)
    ser.write(frcmd)

def set_rate(ser, pump,rate, unit):
    if (unit != 'MH' and unit != 'UH'):
```

```
        raise NameError('You entered an unknown measurement unit. The
allowed values are only \'MH\' and \'UH\'')
    cmd = b'%iRAT%.2f%s\x0D'%(pump,rate,unit.encode())
    output = ser.readline()
    if b'?' in output: print (str(cmd.strip())+' from console not
understood')

def start_pump(ser, pump):
    cmd = b'%iRUN\x0D'%pump # "RUN" command is formatted as a byte string
    ser.write(cmd)

def stop_pump(ser, pump):
    cmd = b'%iSTP\x0D'%pump
    ser.write(cmd)

set_diameter(ser, pump, b'21.00')
set_direction(ser,pump, b'INF')
set_rate(ser, pump, rate, rate_unit)
start_pump(ser, pump)
stop_pump(ser,  pump)
```

The **power supply** was used to power the four series of LEDs and was controlled by Python scripts. The functions from https://github.com/markrages/py_test_interface/blob/master/bk1697.py were used to control the power supply. Below are given several examples of employed functions:

```
def start_pwr_supply(self,volts,amps): # function to start the power
supply configured with the desired configuration

      self.begin_session()

      self.output_on()

      self.set_volts(volts)

      self.set_amps(amps)

def end_session(self): # function to reactivate the physical buttons of
the power supply after an experiment reached its end

        self.ENDS()

# examples of calling the functions above

bk.start_pwr_supply(16.5, 0.3)

bk.end_session()
```

Due to the different electrical properties of the LEDs series, they could not be powered in a simultaneous and controlled manner using a single power supply. Therefore, the four series of LEDs were powered one at a time by turning ON and OFF four **relays** (switches) placed on a board that was connected to the computer and controlled by VirtualHub software or Python. We used functions from Yoctopuce Python library, as exemplified below:

```
def relay1_PULSE(ON_TIME): # function used to light a LED series for the
number of seconds defined by the ON_TIME variable
 bord1_relay1.set_output(YRelay.OUTPUT_ON)
 sleep(ON_TIME)
 bord1_relay1.set_output(YRelay.OUTPUT_OFF)

def relay2_PULSE(ON_TIME):
 bord1_relay2.set_output(YRelay.OUTPUT_ON)
 sleep(ON_TIME)
 bord1_relay2.set_output(YRelay.OUTPUT_OFF)

def relay3_PULSE(ON_TIME):
 bord1_relay3.set_output(YRelay.OUTPUT_ON)
 sleep(ON_TIME)
 bord1_relay3.set_output(YRelay.OUTPUT_OFF)
def relay4_PULSE(ON_TIME):
 bord1_relay4.set_output(YRelay.OUTPUT_ON)
 sleep(ON_TIME)
 bord1_relay4.set_output(YRelay.OUTPUT_OFF)
```

In order to obtain similar photon fluxes from the four LEDs series, different values of forward current are necessary as illustrated in Table 1 in the manuscript. However, the values for the current and voltage set on the power supply could not be modified faster than 1 s, which was limiting during dynamic illumination that required modifications as fast as 0.5 and 1 s. Therefore, we decided to keep the input parameters on the power supply and we introduced the **variable resistors** (potentiometers) in the electrical circuits of the LEDs series emitting at 395 and 443 nm to obtain lower forward currents for those LEDs. The value of resistance for the two series of LEDs was adjusted by adjusting the position of the two potentiometers while measuring the optical output using the cosine corrector and spectrometer. However, we observed that sometimes when the LEDs are operated for periods longer than 6 hours (the maximal duration of the experiments reported in this work), the temperature of the variable resistors increases and the obtained current values are, consequently, modified along with the emitted photon flux. Therefore, for obtaining more stable systems without adjusting the positions of the potentiometers, the method for adjusting the forward current could be improved, maybe by using electronically controlled variable resistors instead of the variable resistors which are mechanically controlled.

The **stepper motor** is used to rotate the LED light source and is driven by an Arduino Uno board. The code to control the stepper motor using Arduino IDE 2.1.1 is shown below:

```
int start;

void setup() {

   Serial.begin(19200);

   Serial.flush();

   pinMode(7, OUTPUT); // Arduino board drives the PUL+ pin of the
driver which has been connected to the Arduino PIN 7
```

```
   pinMode(8, OUTPUT); // Arduino board drives the DIR+ pin of the
driver which has been connected to the Arduino PIN 8

   pinMode(13, OUTPUT); // Arduino board drives the ENA+ pin of the
driver which has been connected to the Arduino PIN 13

   digitalWrite(7, LOW); //

   digitalWrite(8, HIGH); // Set the rotation direction for the
stepper motor HIGH= counter-clockwise

   digitalWrite(13, LOW); // Enable the automatic control of the motor

}

void loop() {

   if(Serial.available()>0){

     start = Serial.read();

     if (start=='1'){

       digitalWrite(8, HIGH); // The motor will rotate counter-
clockwise

       digitalWrite(13, LOW); // Enable the automatic control of the
motor

       delay(1000);  //delayMicroseconds(1000);

       // Driver setting is 1600 pulses-rev, therefore for 90 degrees
it should be 1600/4=400

       for(int i =0;i<400;i++){ // The PWM signal needed to drive the
motor is generated

         digitalWrite(7, HIGH);

         delayMicroseconds(300);

         digitalWrite(7, LOW);

         delayMicroseconds(300);

       }

       digitalWrite(13, HIGH); // Disable the automatic control of the
motor

     }

   else if (start=='2'){

       digitalWrite(8, LOW);

       for(int i =0;i<400;i++){
```

```
          digitalWrite(7, HIGH);
          delayMicroseconds(300);
          digitalWrite(7, LOW);
          delayMicroseconds(300);
        }
      }
    }
}
```

The rotation of the motor is performed by sending commands to the Arduino board via a serial port from PC using Python. In the Arduino code and in the Python script we coded with "1" the 90 degrees counter-clockwise rotations and with "2" the 90 degrees clockwise rotations Below there is an example in Python needed to send the commands to the Arduino board:

```
motor_handle = serial.Serial(Arduino_port,baudrate=19200,timeout=1)
motor_handle.write(b'1')
motor_handle.write(b'2')
```

Dynamic flow and continuous illumination

The concept behind the dynamic flow, as used in the dynamic flow and continuous illumination, is illustrated in Figure S7.

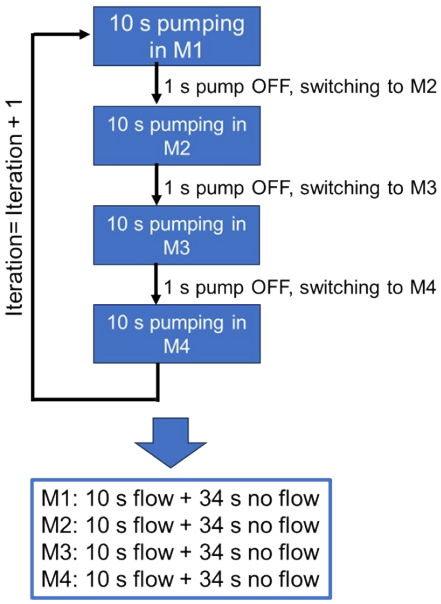

Figure S7 The concept behind the dynamic flow

The dynamic flow, illustrated in Figure S7 and Figure 6a in manuscript, was implemented in Python as follows:

```
for i in range (0, nb_iterations):
    #Start the pump
    start_pump(ser,pump)
    print ("pump is ON")
    #Delay
    time.sleep(10)
    #Stop the pump
    stop_pump(ser, pump)
    print ("pump is stopped")
    time.sleep(34) # in seconds
```

Continuous flow and dynamic illumination

The concept behind the dynamic illumination, as used in the continuous flow and dynamic illumination, is illustrated in Figure S8.

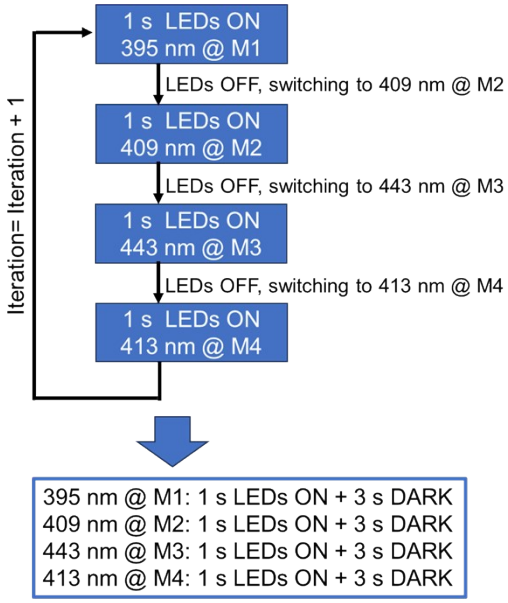

Figure S8 The concept behind the dynamic illumination

The dynamic illumination was implemented in Python as shown below. The number of iterations, x, was calculated in function of the time the LEDs were on, ON_TIME, and the total duration of the experiment:

```
ON_TIME  = 1 # in seconds

for x in range(100):
  relay1_PULSE(ON_TIME)
  relay2_PULSE(ON_TIME)
  relay3_PULSE(ON_TIME)
  relay4_PULSE(ON_TIME)
```

Dynamic flow and dynamic illumination

The flowchart of the dynamic flow and dynamic illumination is illustrated in Figure S9. We used the concept of threading to allow the simultaneous operation and control of the pump and LEDs.
An iteration consists of a full sequence "start-stop" of the pump during which the LEDs are turned on and operated under dynamic illumination. Once the given number of iterations is achieved, the LEDs are turned off via a global variable. This global variable is used by the two threads, for operating the pump and LEDs which are running in parallel. When the thread controlling the pump ends because the given number of iterations is achieved, the global variable is modified in order to transmit that the LEDs should be turned off. The thread in charge of the LEDs will read the modified global variable and the LEDs will stop being powered. This is highlighted in Figure S9 as well by the yellow color.

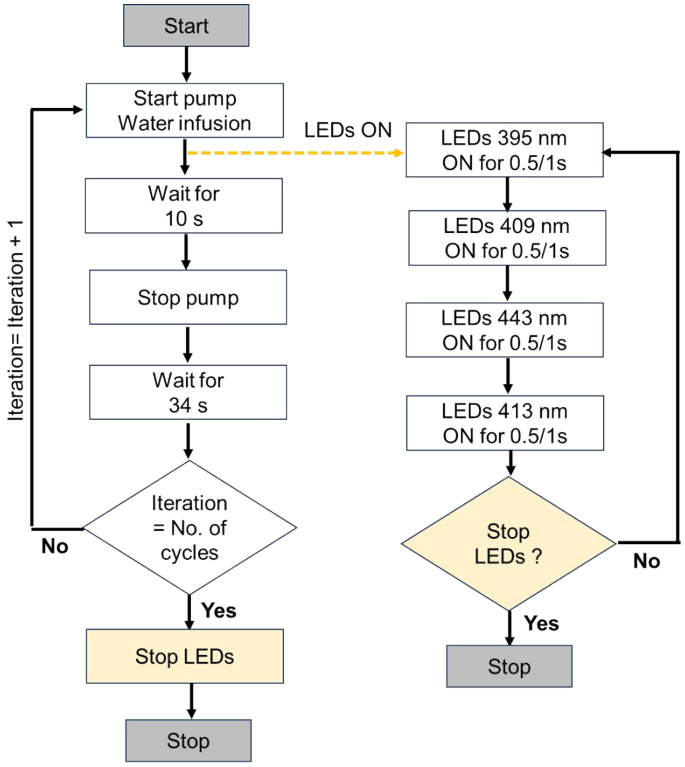

Figure S9 The implemented flowchart of the dynamic pumping and illumination

An example of code for implementing the flowchart illustrated in Figure S9 is shown below:

```
def experiment(pump_port = 'COM9'):
    global LEDs_must_light
    for i in range (0, nb_iterations):
        #Start the pump
        start_pump(ser,pump)
        print ("pump is ON")
        # LEDs are turned on
        LEDs_must_light = 1;
        #Delay
        time.sleep(10)
        #Stop the pump
        stop_pump(ser, pump)
```

```
        print ("pump is stopped")
        time.sleep(34)
    # Turn off the LEDs
    LEDs_must_light = 0
    print("LEDs are turned off")
    LEDs_off()
    print ("the experiment will stop")
    time.sleep(5)
    os._exit(1)

def LEDs_on(ON_TIME=ON_TIME):
    global leds_must_light
    #Start power supply
    bk.start_pwr_supply(volts,amps)
    counter = 9
    das =0
    while 1:
        if counter >1:
            counter  = counter -1
        if leds_must_light:
            if das == 1:
                das = 0
                counter =9
            print("395 nm LEDs ON")
            relay1_PULSE(ON_TIME)
            print("409 nm LEDs ON ")
            relay2_PULSE(ON_TIME)
            print("443 nm LEDs ON")
            relay3_PULSE(ON_TIME)
            print("413 nm LEDs ON")
            relay4_PULSE(ON_TIME)
        else:
            das = 1

def LEDs_off():
    bord1_relay1.set_output(YRelay.OUTPUT_OFF)
    bord1_relay2.set_output(YRelay.OUTPUT_OFF)
    bord1_relay3.set_output(YRelay.OUTPUT_OFF)
    bord1_relay4.set_output(YRelay.OUTPUT_OFF)

if __name__ == "__main__":
    thread1 = threading.Thread(target=LEDs_on, daemon = True)
    thread1.start()
    experiment(pump_port = 'COM9')
```

S3. 2D irradiance distribution for 395 nm LEDs

The irradiance over every point (x, y) on a flat screen at a distance z from the LED array is given by the sum of the irradiances for *N* LEDs:[2,3]

$$E(x,y,z) = z^m A_{LED} L_{LED} \sum_{n=1}^{N} \left\{ \left[x - (N+1-2n)\left(\frac{d}{2}\right) \right]^2 + y^2 + z^2 \right\}^{-\frac{(m+2)}{2}} \quad (1)$$

Where L_{LED} is the radiance (W m^{-2} sr^{-1}) of the LED, A_{LED} is the LED emitting area (m^2), d is the LED-to-LED separation.

The value of *m* for the used LEDs was determined considering the $\theta_{1/2}$ equal to 15 degrees in radian:

$$m = \frac{-ln2}{ln(cos\theta_{1/2})} \cdot 100 \quad (2)$$

To obtain the irradiance maps in mW cm^{-2}, we used the measured irradiance by cosine corrector and spectrometer to fit the value of the product L_{LED} A_{LED}. The irradiance distribution was estimated at different distances LED-photocatalyst ranging between 24 and 54 mm. As can be observed from Figure S10 and Table S1, the ideal distance LED-photocatalyst in terms of mean irradiance, uniformity, and coefficient of variance (CV) would be 34 mm. However, we opted for the distance of 44 mm as it was the smallest achievable one due to the dimensions of the commercial connectors that hold together the metal frame.

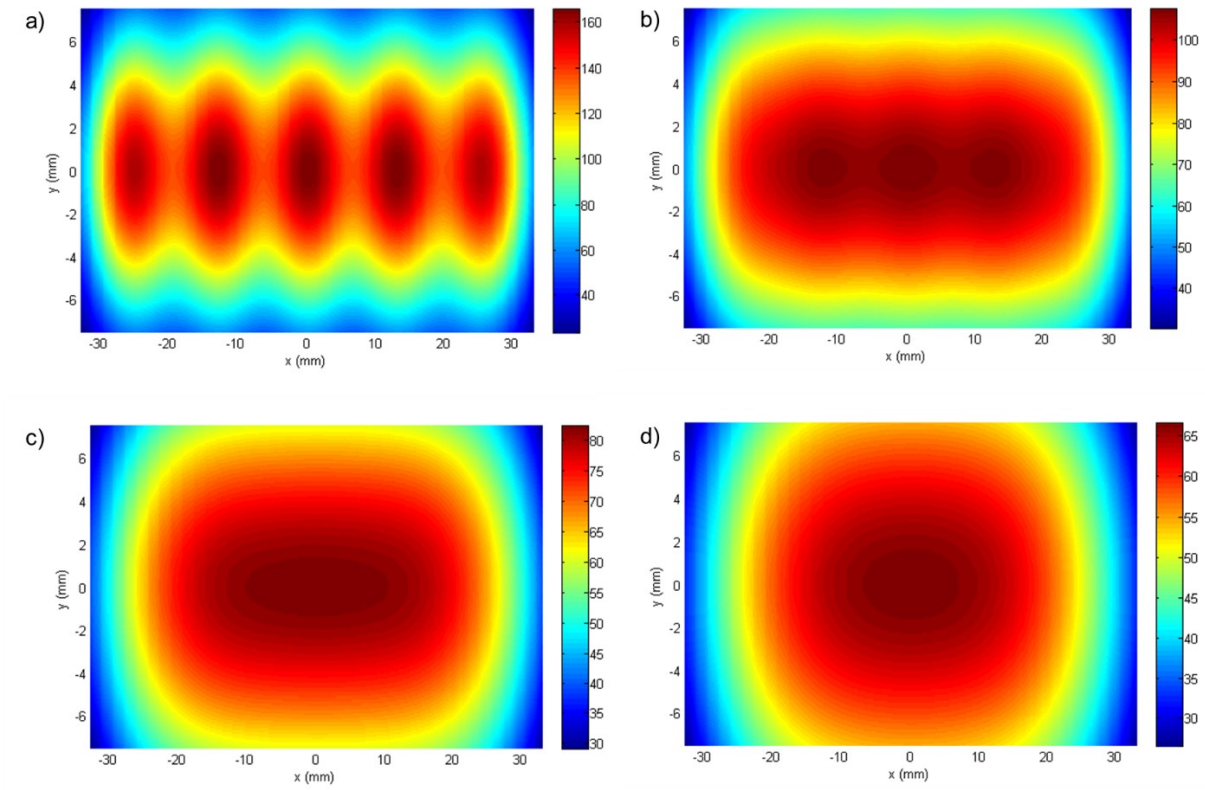

Figure S10. 2D irradiance distribution expressed in mW cm^{-2} for 395 nm LEDs at various distances between the LED tip and photocatalytic film surface a) 24 mm, b) 34 mm, c) 44 mm, and d) 54 mm.

Table S1. Characteristics of the irradiance distribution at different distances between the LED tip and photocatalytic film surface for 395 nm LEDs. STD represents the standard deviation and CV is the coefficient of variance.

Distance (mm)	Mean Irradiance (mW cm^{-2})	Max Irradiance (mW cm^{-2})	Min irradiance (mW cm^{-2})	STD (mW cm^{-2})	CV	Uniformity (%)
24	142.2	165.4	60.7	23.1	0.162	86
34	96.8	107.5	49.5	15.5	0.160	90
44	**72.3**	**82.3**	**39.3**	**12.6**	**0.174**	**88**
54	57.0	66.6	32.4	10.3	0.182	86

The uniformity parameters were calculated as follows:

$$Uniformity = \frac{Mean\ irradiance}{Max\ irradiance} \cdot 100 \quad (3)$$

$$CV = \frac{STD}{Mean\ irradiance} \quad (4)$$

S4. Photonic efficiency comparison

The photonic efficiency can be calculated as follows:[4]

$$\xi = \frac{rate\ of\ reaction}{incident\ photon\ flux} \quad (5)$$

As we considered only the IMD degradation efficiencies below 30 %, we estimate the average reaction rate as:[4]

$$Rate\ of\ reaction = \frac{\Delta C_{IMD} V}{\Delta t} \quad (6)$$

Where ΔC_{IMD} is the change in concentration of imidacloprid in the time interval Δt and V is the liquid volume in the microreactor channel.

Next, to compare the photonic efficiencies of the continuous and dynamic illumination we defined the following ratio:

$$\frac{\xi_{continuous}}{\xi_{dynamic}} = \frac{(\Delta C_{IMD} V)_{\ continuous}}{incident\ photon\ flux} \cdot \frac{incident\ photon\ flux\ \cdot\ duty\ cycle}{(\Delta C_{IMD} V)_{\ dynamic}} = \frac{(\Delta C_{IMD} V)_{\ continuous} \cdot\ duty\ cycle}{(\Delta C_{IMD} V)_{\ dynamic}} \quad (7)$$

Considering the degradation efficiencies of 30.2% for continuous illumination and 15.7% for the dynamic illumination, a duty cycle of 25%, we found that the photonic efficiency ratio, $\frac{\xi_{continuous}}{\xi_{dynamic}}$, is equal to 1.9.

S5. References

1 R. Garg, R. Gupta, and A. Bansal, Int. J. Environ. Sci. Technol. 2021, **18**, 1425.

2 I. Moreno, M. Avendaño-Alejo, and R. I. Tzonchev, Appl Opt, 2006, **45**, 2265-2272.

3 A. Roibu, R. B. Morthala, M. E. Leblebici, D. Koziej, T. Van Gerven, and S. Kuhn, React Chem Eng, 2018, **3**, 849-865.

4 Y. Ku, S.-J. Shiu and H-C. Wu, J. Photochem. Photobiol. A, 2017, **332**, 299–305.