Supplementary material

Arduino Uno© is an open-source electronic platform comprising both hardware and software. The hardware is based on the 8-bit ATMEGA328 microcontroller. The Arduino features 14 digital pins that can be utilized as input and/or output, and six analog reading channels (A0...A5). Additionally, it includes a USB connection port, facilitating communication with any microcomputer.

The Integrated Development Environment (IDE) provides a user-friendly programming interface using the C++ language. The automation of the proposed experiment, which integrates photogate, enables us to conduct various analyses of free fall movement. Details regarding construction of the photogate are presented in Figure S1(a)-(b)

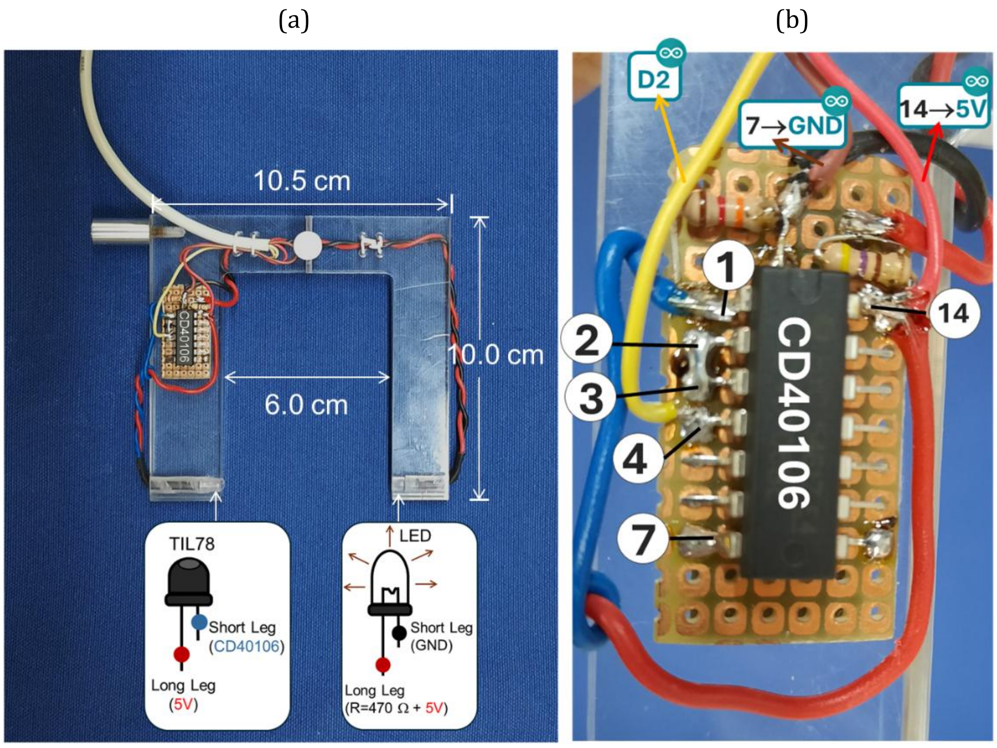


Figure S1. (a) Photogate constructed using a U-shaped acrylic profile, showing the geometric dimensions and the connections for the IR LED and phototransistor (TIL78). (b) The connections of the two Schmitt Trigger (CD40106) ports, which are in series, are highlighted. The red (14), brown (7), and yellow wires that must be connected to the 5V, GND, and D2 pins of the Arduino, respectively, are also indicated.

(a) Electrical circuit

Figure S2(a) depicts the electrical diagram employed to analyze the relationship between time in the free fall. Although this diagram has been previously introduced and discussed within the article, we aim to highlight specific aspects and propose alternative options for potential replacements, if deemed necessary.

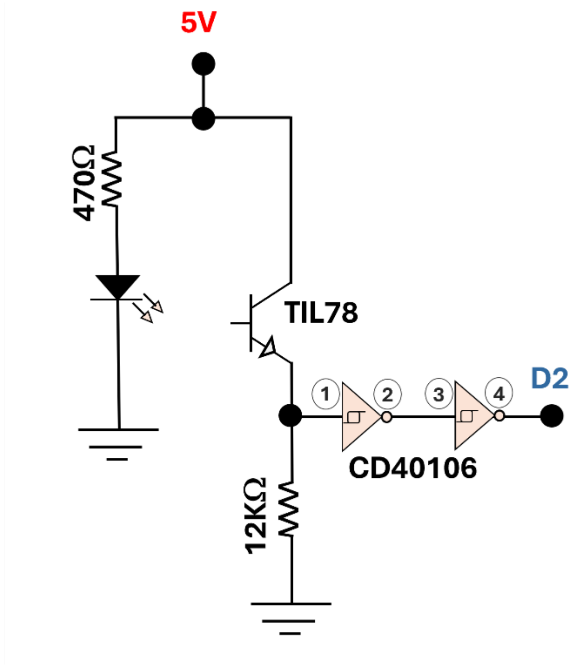


Figure S2. The photogate electrical diagram. The point labeled as 4, corresponding to the output of the CD40106 Schmitt Trigger pair, should be connected to Arduino pin D2.

The CD40106 integrated circuit contains six independent inverting logic gates of the Schmitt Trigger type. In this work, we use a pair of Schmitt Triggers connected between pins 1 and 4, ensuring that the logic state at the input (High or Low) on pin 1 is mirrored at the output on pin 4. This configuration facilitates understanding the photogate's operation—specifically, when the phototransistor detects light (High) and when the LED is blocked, preventing light from reaching the phototransistor, resulting in a "Low" state.

For users opting to work with inverted logic, it is possible to use only one gate of the CD40106. However, this would require corresponding adjustments to the program logic.

(b) Code developed

The script for collecting nineteen data points for each flag throw is shown in Figure S3, along with some additional comments. The Arduino program (*FreeFall.ino*) is available for download at https://drive.google.com/file/d/1Y61jkcduGSxRV6o0jvTlapQS4uK4vnBq/view?usp=sharing .

```
//***********HARDWARE DEFINITIONS************************
#define LED  13
#define interruptPin  2

//*************VARIABLES DEFINITIONS******************
int option, i, ii = 0, j = 0, jj;
float T[100];


void setup()
{
    Serial.begin(9600);
    pinMode(LED, OUTPUT);
    Serial.println("LABEL,t(s),S(m),t(s),S(m)");
    // attachInterrupt(0, Time, CHANGE); //Activating
external interrupt mode (Pin D2)
    attachInterrupt(digitalPinToInterrupt(interruptPin),
Time, CHANGE);
}

void Time()// Function to capture time in microseconds
using external interrupt
{
    T[j] = micros();
    j = j + 1;
}
void loop()
{
    if (Serial.available() > 0)
    {
        option = Serial.read();
    }

    switch(option)
```

```
{
                case 'A':
// Sending data to Excel
digitalWrite(LED, HIGH);
Serial.println("LABEL,t(s),S(m),t(s),S(m)");

            jj=2;     // Row control
for(i=0; i<19; i++)
                    {
Serial.print("CELL,SET,");
Serial.print ('A');  // Column A
Serial.print (jj);  // Row
Serial.print(",");
Serial.println((T[i]-T[0])/1000000,4); // Time
in seconds
Serial.print("CELL,SET,");
Serial.print ('B'); // Column B
Serial.print (jj);
Serial.print(",");
Serial.println(i*0.0192,4); // Distance in
meters
jj=jj+1; // Advancing to the next row
}
//Clearing the time storage array after
sending
j=0;
for(i=0; i<30; i++)
       {
T[i]=0;
       }
digitalWrite(LED,LOW);
option=100;
break;
```

```
          case 'B':
// Clearing columns A and B
jj=2;
Serial.println("RESETROW");
digitalWrite(LED, HIGH);

for(i=0; i<19; i++)
                    {
Serial.print("CELL,SET,");
Serial.print ('A');  // Column A
Serial.print (jj);   // Row
Serial.print(",");
Serial.println(0);
Serial.print("CELL,SET,");
Serial.print ('B'); // Column
B
Serial.print (jj);
Serial.print(",");
Serial.println(0);

jj=jj+1; //Advancing to the next
row
                    }
digitalWrite(LED, LOW);
option=100; // Resetting the
option
break;
```

Figure S3. The Arduino program.

The program begins by defining hardware components and variables. The LED is connected to pin 13, while the photogate is connected to pin 2 of the Arduino, and an array T[] is defined to store the time measurements during data collection.

In the *setup()* function, serial communication is initialized at a baud rate of 9600.

Pin D2 is configured as an external interrupt pin, triggering the *Time* function upon any change in its state.

The pin connected to the interrupt (pin 2) is set up to trigger an interrupt on any change (from HIGH to LOW or LOW to HIGH), activating the *Time()* function. This function will capture and store time when triggered, marking key transitions (from light to dark or vice versa) as the flag falls.

The main program logic occurs in the loop() function, which continuously checks the serial port for any available data. When data is received (A, B, C, D), it determines the action to take based on the value of the variable *option*, using a switch-case structure:

Data Acquisition (via Interrupts): The data acquisition is handled separately from the switch-case structure. As the flag falls and the photogate detects changes in transparency (transitions between light and dark), the *attachInterrupt()* command triggers the *Time()* function. The function reads the Arduino's internal clock (via *the micros()* function) and stores the recorded time in the *T[]* array. This process continues automatically as the flag passes through the photogate, storing up to 19-time values.

Case 'A': When the string 'A' is received, the program enters this case. First, the LED is turned on to signal the start of data processing. The program then sends the previously recorded time (from the T[] array) and distance values (from the *T[]* array) to the Excel spreadsheet via the PLX-DAQ interface. The times are displayed in column 'A', while the corresponding distances are displayed in column 'B'.

Case 'B': This case resets the table in Excel by writing '0' to all cells in columns 'A' and 'B'. It effectively clears any previous data to prepare the table for a new experiment. The LED is toggled to signal this reset process.

Case 'C': Similar to *Case 'A'*, this case sends the data from the T[] array to Excel. However, the results are sent to columns 'C' (time) and 'D' (distance), allowing for a comparison between two separate data sets. This could be useful, for instance, when comparing the fall times of objects with different masses.

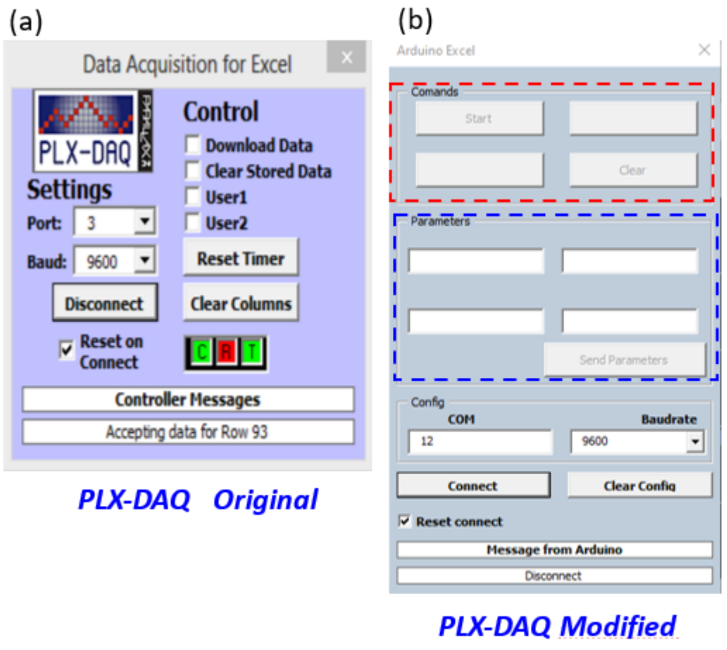
Case 'D': This case resets columns 'C' and 'D', similar to how Case 'B' resets columns 'A' and 'B'. This allows for a new set of data to be collected in columns 'C' and 'D', without affecting data in columns 'A' and 'B'.

In summary, the program allows for the collection of fall times during the flag's descent and their direct transmission to an Excel sheet. This provides a straightforward and automated way to analyze the time and distance data for different experimental conditions, such as varying masses, while also offering functionality to reset or repeat measurements as needed.

(c) Experiment Execution and Data Analysis

To enhance user accessibility and facilitate the dissemination of the developed module, we employed Microsoft Excel for data storage, graphical presentation of results, and potential mathematical analyses, such as curve fitting. To further improve accessibility, we modified the original PLX-DAQ software, for seamless integration with Excel. Figure S4(a) displays the control box of the original PLX-DAQ, while Figure S4(b) shows the modified control box tailored to our specific application.

In Figure S4(a), it is evident that the original PLX-DAQ allows users to select the communication COM port, adjust the communication speed, and connect or disconnect the Arduino from the computer. However, it lacks functionality for initiating data collection or controlling experimental parameters. For example, users cannot specify the number of data points to be collected or the number of measurements to be performed. Our modifications address these limitations, providing greater control over the experimental setup and data acquisition process.



Port: 3
Baud: 9600
Accepting data for Row 93

COM 12
Baudrate 9600

Figure S4. (a) Image of the original PLX-DAQ program developed by Parallax (b) Image of the modified PLX-DAQ with the necessary changes for *s-t* curve acquisition

In the modified control box, we retained all the original functionalities of the PLX-DAQ while adding several new features: four buttons capable of sending characters (e.g., "A", "B", "C", "D") to the microcontroller, as indicated by the dashed red rectangle in Figure S4(b), and four input boxes that allow the transmission of numerical parameters to the Arduino, highlighted by the dashed blue rectangle in the same figure.

To simplify usage and avoid the need for users to install Excel macros manually, we provided a preconfigured spreadsheet (FreeFall.xlsm) with all necessary macros embedded. This file is available at https://drive.google.com/file/d/1Y61jkcduGSxRV6o0jvTlapQS4uK4vnBq/view?usp=sharing . If you wish to implement modifications to the add-on, you can do so from there. When saving collected data, remember to save it with the same file extension (xlsm).

Additionally, there is a settings sheet where users can adjust button labels and parameter boxes to suit different applications, making customization more intuitive. As shown in Figure S4, pressing "*Ctrl+Q*" will bring up the control box on your monitor for easy access.

Finally, we would like to mention some precautions for properly conducting the experiment. The distances of the light (L = 19.54 mm) and dark (L = 19.54 mm) bands are parameters defined in the program and are measured under the assumption that the flag crosses the photogate perpendicularly, which is an important point to note. If the launch is performed with a relative inclination between the flag and the photogate, as shown in Figure S5(a), the measured values of gravitational acceleration will have significant errors. Therefore, care must be taken to ensure that the launch is performed as depicted in Figure S5(b).

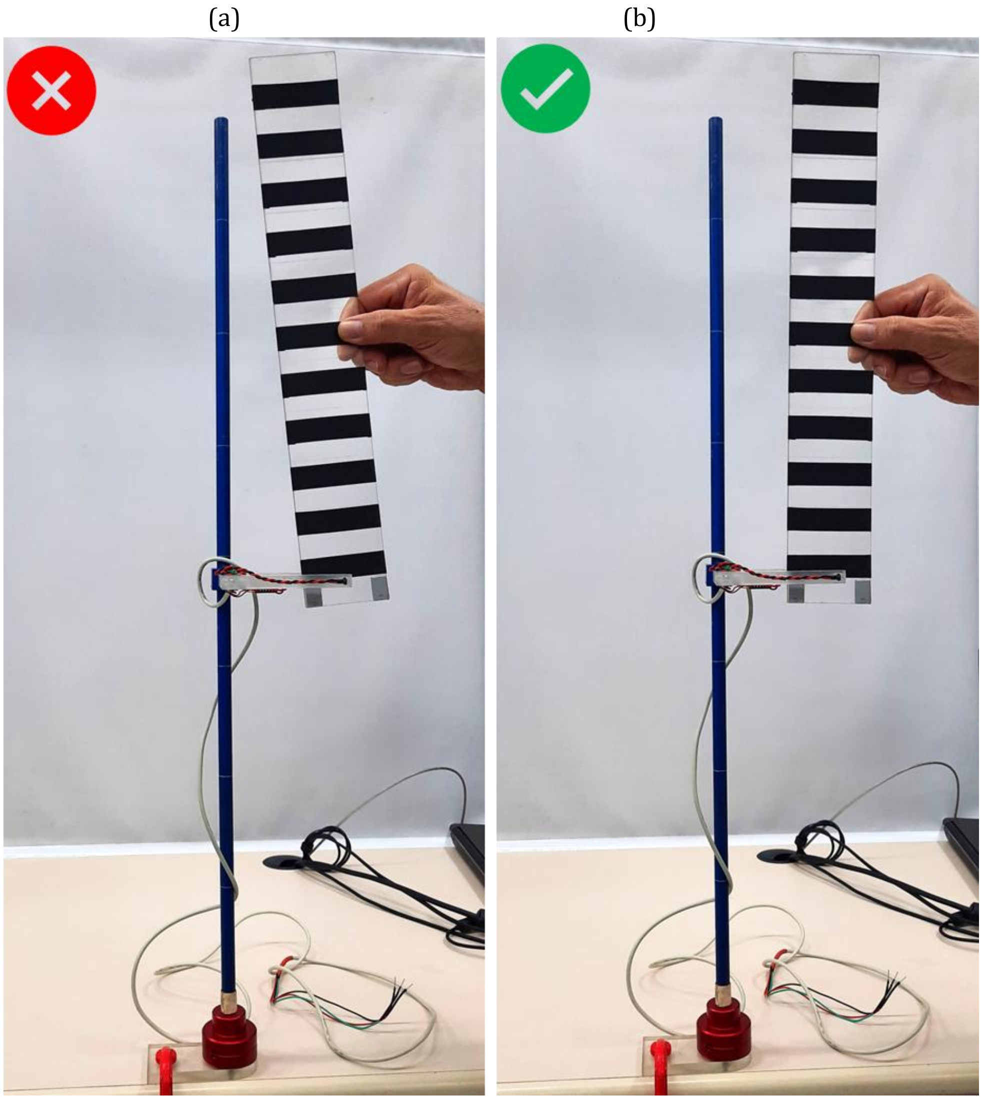


Figure S5. Examples of launches. In (a), an incorrect launch is shown, where the flag has an inclination angle different from 90º relative to the photogate. In (b), the launch is performed under the correct conditions, with the flag and photogate perpendicular to each other.