Interfacing the USB3 Tau2 module from Workswell using Python's ctypes library

1. Introduction to the Project

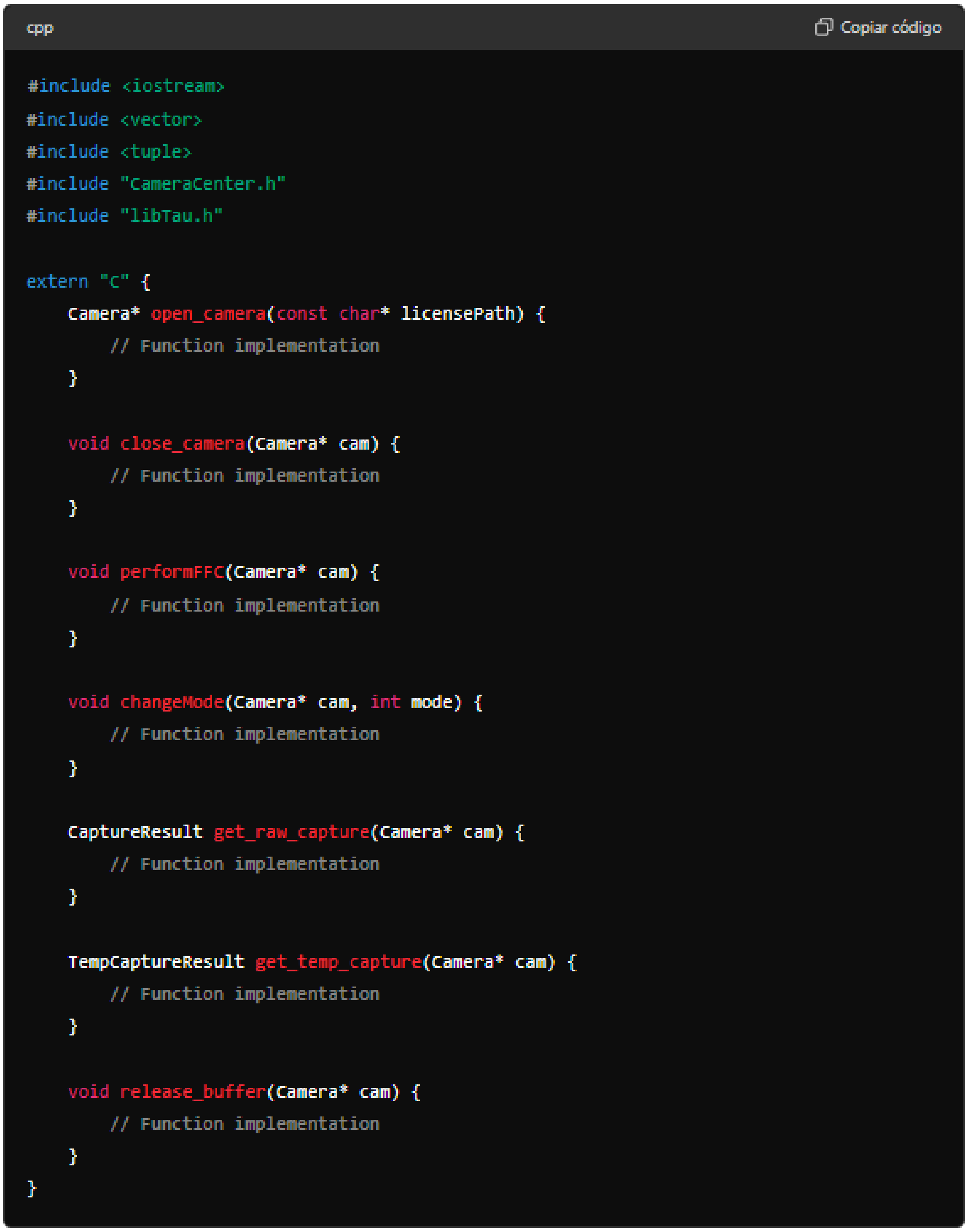
Create a C++ library for camera operations and interact with it using Python's ctypes library.

2. Setting up the environment

- **Build-essential package:** the build-essential package contains an informational list of packages which are considered essential for building Debian packages, including the GNU C compiler, g++, and make utility.
- **Python:** make sure Python is installed. In this project you will need to use the ctypes library, that is already included with Python's standard library. Once Python is installed, you can use ctypes directly in your scripts.
  - It is helpful to install all your packages in virtual environments.
- **Additional Dependencies:**
  - Workswell WIC SDK and Pleora eBUS SDK (they can be installed using the executable installer included in this repository).

3. Writing the C++ Code

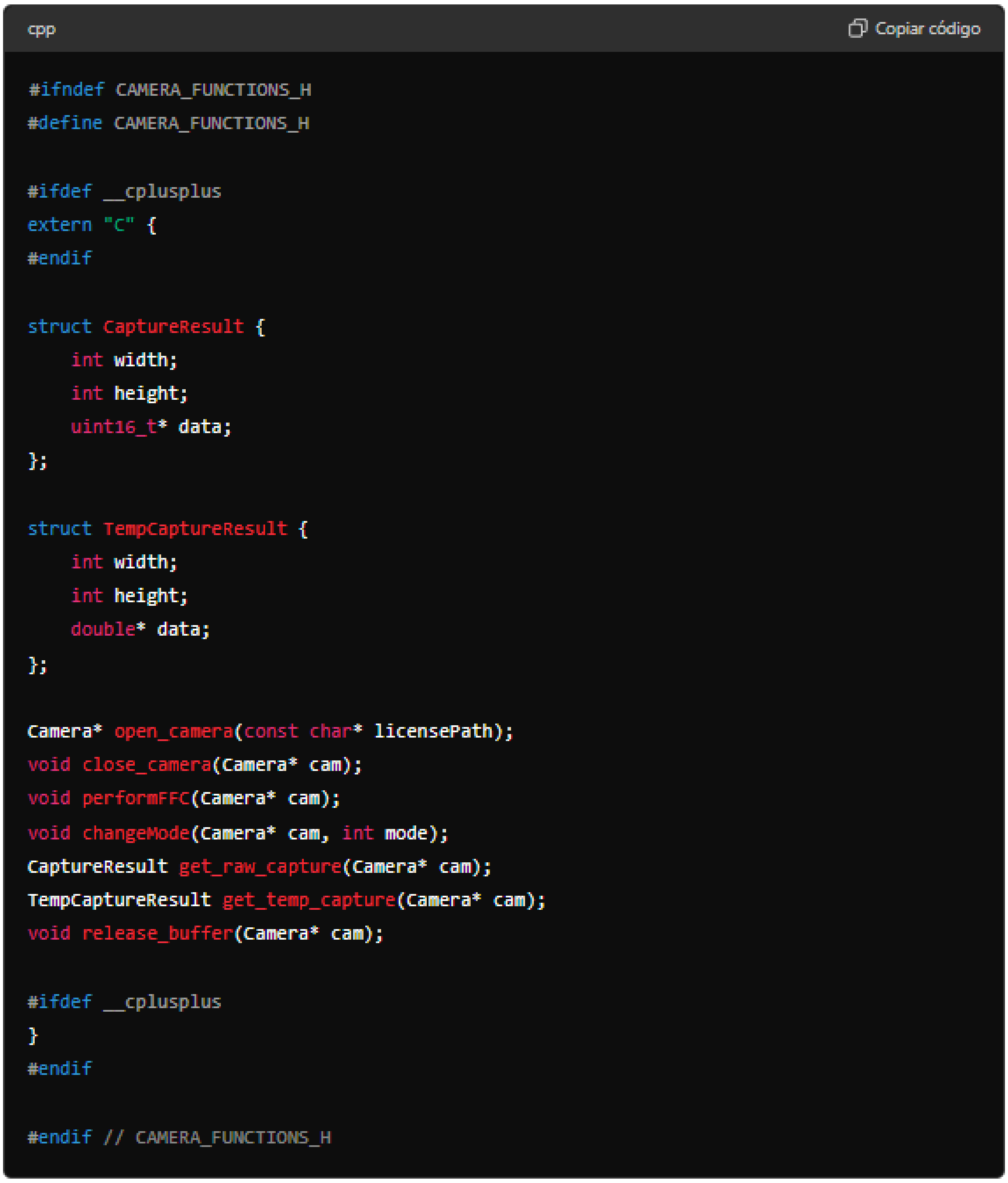
- **libTau.cpp:** in the code you will have something like the snapshot represented in ***Figure 1***.
  - By using extern "C", you ensure that the functions have C linkage. This means that they can be called from C code or from other languages that can interface with C, such as Python via ctypes. Without extern "C", the Python ctypes library would not be able to find the functions in the shared library because it would look for the mangled names instead of the plain C names. This approach ensures that the C++ functions you want to call from Python using ctypes have predictable, unmingled names.



```cpp
#include <iostream>
#include <vector>
#include <tuple>
#include "CameraCenter.h"
#include "libTau.h"

extern "C" {
    Camera* open_camera(const char* licensePath) {
        // Function implementation
    }

    void close_camera(Camera* cam) {
        // Function implementation
    }

    void performFFC(Camera* cam) {
        // Function implementation
    }

    void changeMode(Camera* cam, int mode) {
        // Function implementation
    }

    CaptureResult get_raw_capture(Camera* cam) {
        // Function implementation
    }

    TempCaptureResult get_temp_capture(Camera* cam) {
        // Function implementation
    }

    void release_buffer(Camera* cam) {
        // Function implementation
    }
}
```

Figure 1. Main code to interact with the camera

- **libTau.h:** the header file has declared all the methods we are going to use to interact with the camera, as seen in ***Figure 2***. It is imported in the .cpp file.



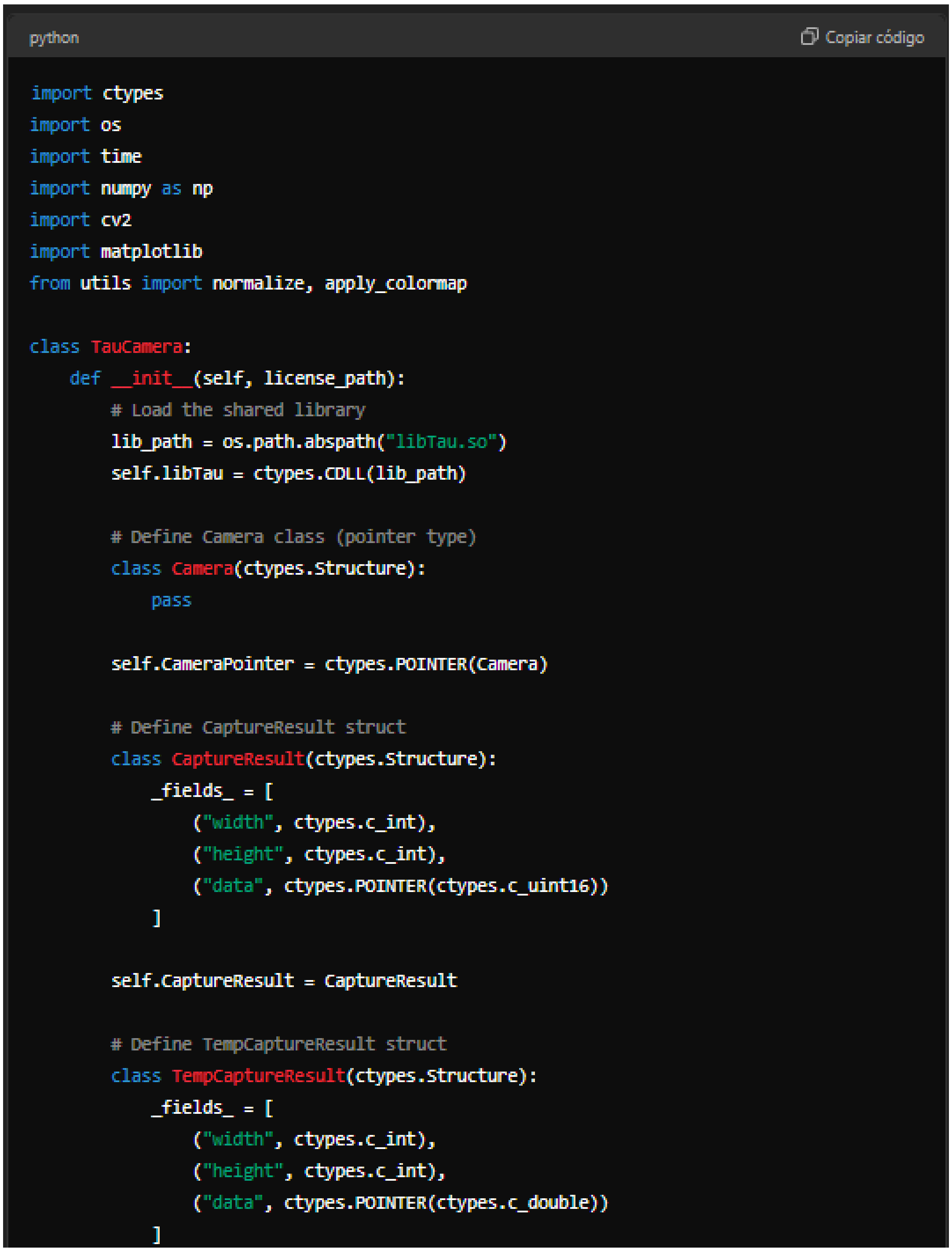
```cpp
#ifndef CAMERA_FUNCTIONS_H
#define CAMERA_FUNCTIONS_H

#ifdef __cplusplus
extern "C" {
#endif

struct CaptureResult {
    int width;
    int height;
    uint16_t* data;
};

struct TempCaptureResult {
    int width;
    int height;
    double* data;
};

Camera* open_camera(const char* licensePath);
void close_camera(Camera* cam);
void performFFC(Camera* cam);
void changeMode(Camera* cam, int mode);
CaptureResult get_raw_capture(Camera* cam);
TempCaptureResult get_temp_capture(Camera* cam);
void release_buffer(Camera* cam);

#ifdef __cplusplus
}
#endif

#endif // CAMERA_FUNCTIONS_H
```

Figure 2. Header file to interact with the camera

- **Now, use "source INFO_ENV_VARS" to export all libraries paths or put it in the ~/bashrc file.**
- **Finally, compile the code using the Makefile you have in the repository and use the python example code provide in the repository.**

4. Python code explanation

The ctypes library provides C compatible data types and allows calling functions in DLLs or shared libraries. This process is referenced in in ***Figure 3, Figure 4*** and ***Figure 5***.

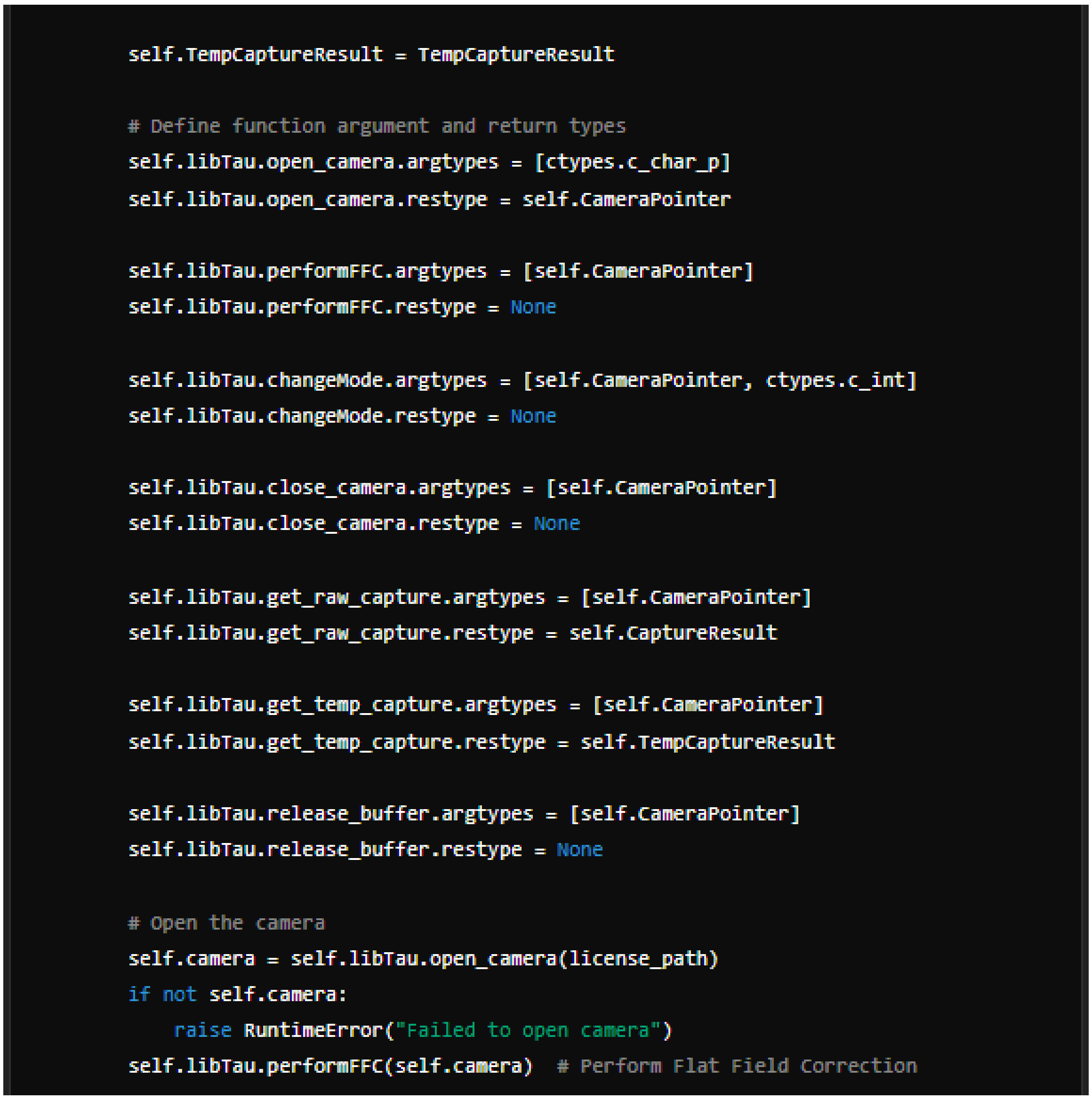


python   Copiar código

```python
import ctypes
import os
import time
import numpy as np
import cv2
import matplotlib
from utils import normalize, apply_colormap

class TauCamera:
    def __init__(self, license_path):
        # Load the shared library
        lib_path = os.path.abspath("libTau.so")
        self.libTau = ctypes.CDLL(lib_path)

        # Define Camera class (pointer type)
        class Camera(ctypes.Structure):
            pass

        self.CameraPointer = ctypes.POINTER(Camera)

        # Define CaptureResult struct
        class CaptureResult(ctypes.Structure):
            _fields_ = [
                ("width", ctypes.c_int),
                ("height", ctypes.c_int),
                ("data", ctypes.POINTER(ctypes.c_uint16))
            ]

        self.CaptureResult = CaptureResult

        # Define TempCaptureResult struct
        class TempCaptureResult(ctypes.Structure):
            _fields_ = [
                ("width", ctypes.c_int),
                ("height", ctypes.c_int),
                ("data", ctypes.POINTER(ctypes.c_double))
            ]
```

Figure 3. Tau camera class (I)

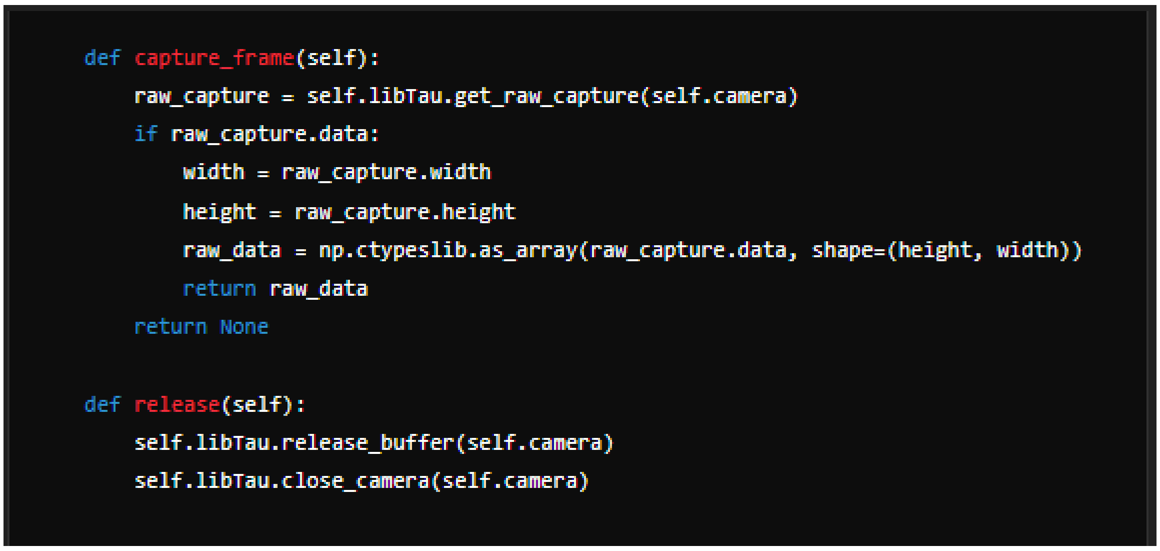
- **Class Initialization**: The TauCamera class initializes the camera by loading the shared library, defining the necessary data structures, and setting up the function prototypes.
- **Loading the Shared Library**: ctypes.CDLL(lib_path) loads the C shared library (libTau.so) into the Python script, allowing access to its functions and data structures.
- **Defining C-compatible Data Types**: The ctypes.Structure classes (Camera, CaptureResult, TempCaptureResult) define C-compatible structures. This ensures that Python can correctly interpret the data from the C library.



```python
        self.TempCaptureResult = TempCaptureResult

        # Define function argument and return types
        self.libTau.open_camera.argtypes = [ctypes.c_char_p]
        self.libTau.open_camera.restype = self.CameraPointer

        self.libTau.performFFC.argtypes = [self.CameraPointer]
        self.libTau.performFFC.restype = None

        self.libTau.changeMode.argtypes = [self.CameraPointer, ctypes.c_int]
        self.libTau.changeMode.restype = None

        self.libTau.close_camera.argtypes = [self.CameraPointer]
        self.libTau.close_camera.restype = None

        self.libTau.get_raw_capture.argtypes = [self.CameraPointer]
        self.libTau.get_raw_capture.restype = self.CaptureResult

        self.libTau.get_temp_capture.argtypes = [self.CameraPointer]
        self.libTau.get_temp_capture.restype = self.TempCaptureResult

        self.libTau.release_buffer.argtypes = [self.CameraPointer]
        self.libTau.release_buffer.restype = None

        # Open the camera
        self.camera = self.libTau.open_camera(license_path)
        if not self.camera:
            raise RuntimeError("Failed to open camera")
        self.libTau.performFFC(self.camera)  # Perform Flat Field Correction
```

- *Figure 4. Tau camera class (II)*

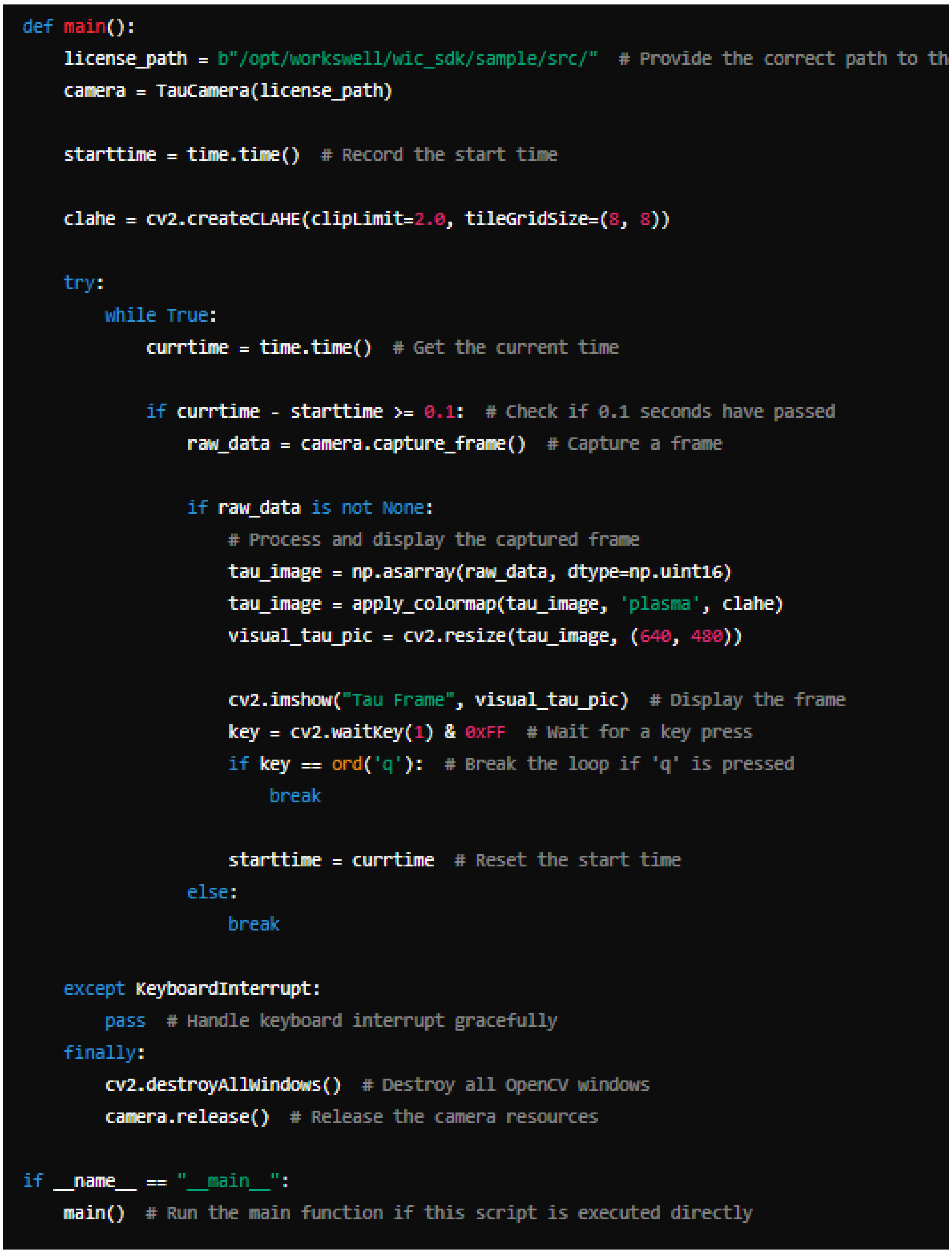
- **Function Prototypes**: The argtypes and restype attributes define the argument and return types of the C functions. This ensures that ctypes correctly handles the data being passed to and from the C library functions.



```python
def capture_frame(self):
    raw_capture = self.libTau.get_raw_capture(self.camera)
    if raw_capture.data:
        width = raw_capture.width
        height = raw_capture.height
        raw_data = np.ctypeslib.as_array(raw_capture.data, shape=(height, width))
        return raw_data
    return None

def release(self):
    self.libTau.release_buffer(self.camera)
    self.libTau.close_camera(self.camera)
```

Figure 5. Tau camera class (III)

- **Frame Capture**: The capture_frame method captures a frame from the camera and converts the raw data into a numpy array.



```python
def main():
    license_path = b"/opt/workswell/wic_sdk/sample/src/"  # Provide the correct path to th
    camera = TauCamera(license_path)

    starttime = time.time()  # Record the start time

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

    try:
        while True:
            currtime = time.time()  # Get the current time

            if currtime - starttime >= 0.1:  # Check if 0.1 seconds have passed
                raw_data = camera.capture_frame()  # Capture a frame

                if raw_data is not None:
                    # Process and display the captured frame
                    tau_image = np.asarray(raw_data, dtype=np.uint16)
                    tau_image = apply_colormap(tau_image, 'plasma', clahe)
                    visual_tau_pic = cv2.resize(tau_image, (640, 480))

                    cv2.imshow("Tau Frame", visual_tau_pic)  # Display the frame
                    key = cv2.waitKey(1) & 0xFF  # Wait for a key press
                    if key == ord('q'):  # Break the loop if 'q' is pressed
                        break

                    starttime = currtime  # Reset the start time
                else:
                    break

    except KeyboardInterrupt:
        pass  # Handle keyboard interrupt gracefully
    finally:
        cv2.destroyAllWindows()  # Destroy all OpenCV windows
        camera.release()  # Release the camera resources

if __name__ == "__main__":
    main()  # Run the main function if this script is executed directly
```

Figure 6. Main program

- **Resource Management**: The release method ensures that resources are properly released by calling the appropriate functions from the shared library.
- **Main Loop**: The main function initializes the TauCamera object and continuously captures and displays frames from the camera until interrupted by the user.