**FCV Project Documentation**

**1. Framework Design**

**1.1 Architecture Overview**

The framework consists of three main components:

* Directory Selection Interface (GUI)
* Configuration Parser
* Augmentation Engine

The system is made up of atomic parts, each responsible for a specific type of change (augmentation). This makes it easy to add new changes or modify existing ones.

**1.2 Core Components**

* Directory Selection: Implemented using *tkinter* for cross-platform compatibility
* Configuration Parser: Robust text-based parser with error handling
* Augmentation Engine: Supports both single operations and operation chains
* Output Management: Automated directory creation and systematic file naming

**2. Configuration File Structure**

**2.1 Basic Syntax**

* Each line represents either a single operation or a chain of operations
* Format: operation1 parameter1 [operation2 parameter2 ...]
* Lines starting with '#' are treated as comments
* Empty lines are ignored

**2.2 Configurations Examples**

**A screenshot of a computer program

Description automatically generated**

**3. List of Implemented Algorithms**

1. **Gamma Correction**

* Uses the power-law transform:
* Pre-computes a lookup table (0-255) for efficiency and applies it using *cv2.LUT*
* Gamma > 1 darkens; Gamma < 1 brightens the image.

1. **Brightness Adjustment**

* Implements a pixel-by-pixel operation without using OpenCV functions
* Adds the brightness value directly to each color channel
* Uses np.clip to ensure values stay within valid range [0, 255]
* Handles each color channel (RGB) separately

1. **Box Blur**

* Utilizes *cv2.blur* which implements a simple averaging filter
* Kernel size must be an odd number to have a well-defined center pixel
* Each pixel becomes the average of its neighbors within the kernel

1. **Contrast Adjustment**

* Uses *cv2.convertScaleAbs* for efficient implementation
* Formula:
* Alpha parameter controls contrast multiplication factor
* Beta parameter (set to 0) could add brightness offset if needed

1. **Rotation**

* Calculates new dimensions using trigonometry to fit rotated image
* Uses *cv2.getRotationMatrix2D* to get the transformation matrix
* Adjusts the transformation matrix to center the rotated image
* Applies the transformation using *cv2.warpAffine* with bilinear interpolation

1. **Scaling**

* Uses *cv2.resize* for efficient implementation
* Calculates new dimensions based on scale factor
* Maintains aspect ratio by applying same scale factor to both dimensions

1. **Shearing**

* Creates a 2x3 transformation matrix for shearing
* Applies horizontal shear while keeping vertical positions unchanged
* Uses *cv2.warpAffine* for the actual transformation, keeping the original image dimensions

1. **Flipping**

* Reorders pixels manually, without OpenCV functions, for horizontal (1), vertical (0), or both (-1) flips.
* Creates a new array and reverses pixels according to the flip mode.
* Separate logic handles each flip type.