Detailed flow of operations:

1. The program starts by including necessary libraries and namespaces.

2. The `cartoonise` function is defined, which takes a `Mat` object (representing a frame) as input and applies a cartoon-like effect to it.

3. Inside the `main` function:

   a. The OpenCV version is printed.

   b. A `VideoCapture` object named `cap` is created to read the input video file "Hackathon.mp4".

   c. If the video file fails to open, an error message is printed, and the program exits with an error code.

   d. The frame rate (fps) and frame size of the input video are obtained.

   e. A `VideoWriter` object named `writer` is created to write the output video file "Hackathon_output2.mp4" with the same frame rate and size as the input video.

   f. If the video writer fails to open, an error message is printed, and the program exits with an error code.

   g. Two `Mat` objects, `image1` and `image2`, are created to store consecutive frames from the input video.

   h. The first frame from the input video is read into `image1`.

   i. A loop is started to process each pair of consecutive frames:

      i. The current frame (`image1`) is converted to grayscale (`gray1`).

      ii. The next frame (`image2`) is read from the input video and converted to grayscale (`gray2`).

      iii. Absolute difference between `gray1` and `gray2` is calculated and stored in `gotFrame`.

      iv. Thresholding is applied to `gotFrame` to obtain a binary image (`thresh`).

      v. Dilation operation is performed on the binary image to enhance the detected objects (`dilate`).

      vi. Contours of detected objects are found in the dilated image.

      vii. Contours are drawn on `image1` using green color.

      viii. The processed frame (`image1`) is written to the output video file.

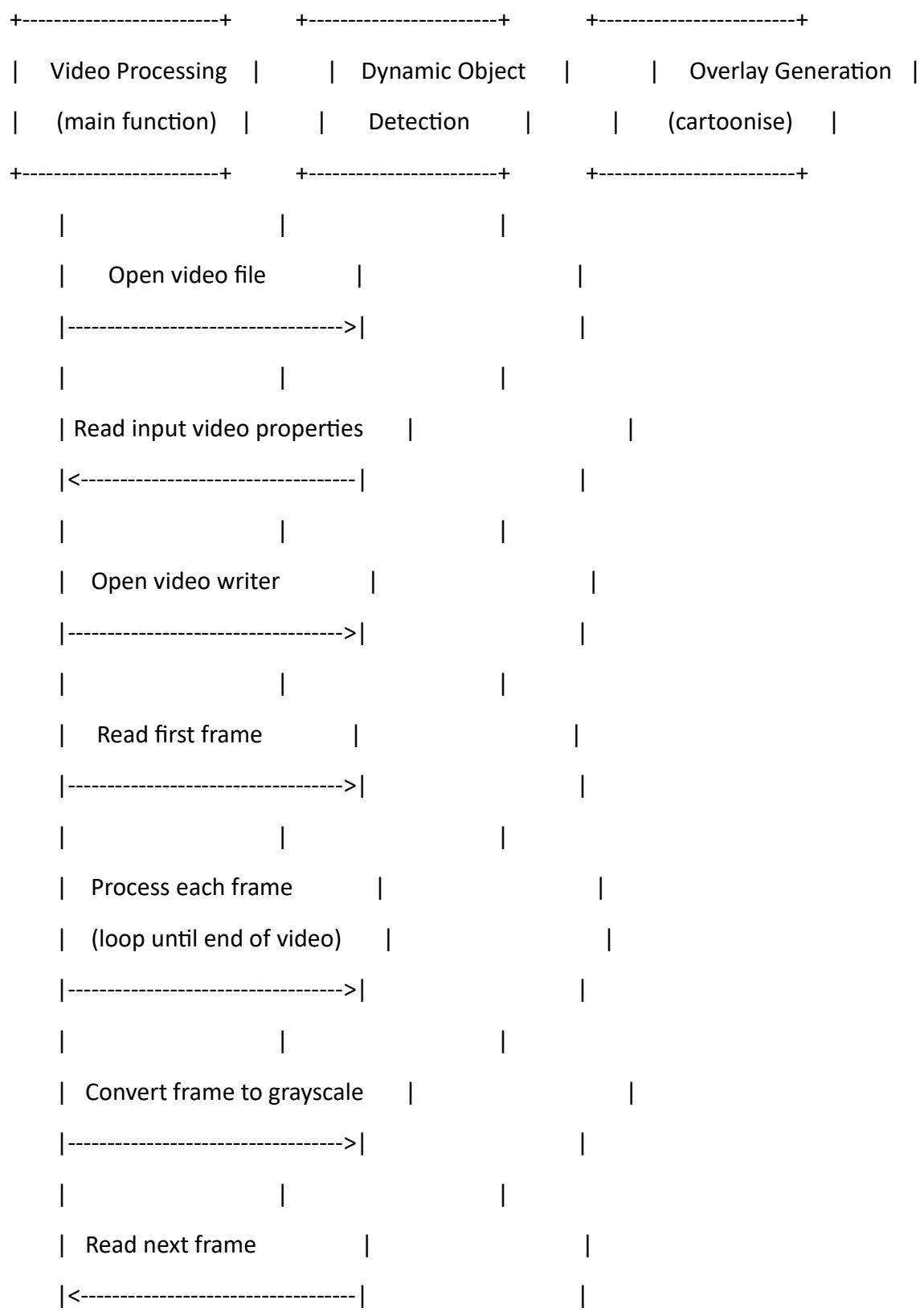
      ix. The next frame from the input video is read into `image1`.

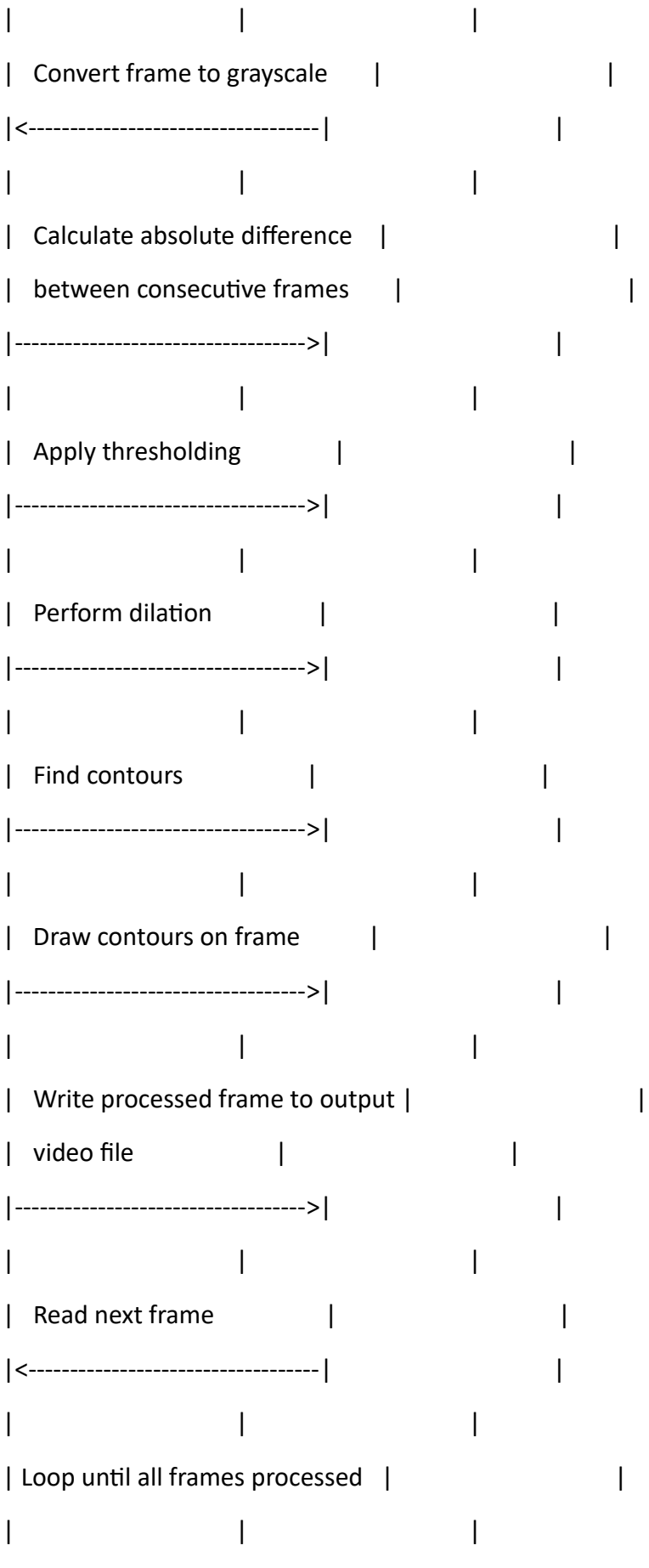
   j. Once all frames are processed, the video writer is released, and the input video capture object is released.

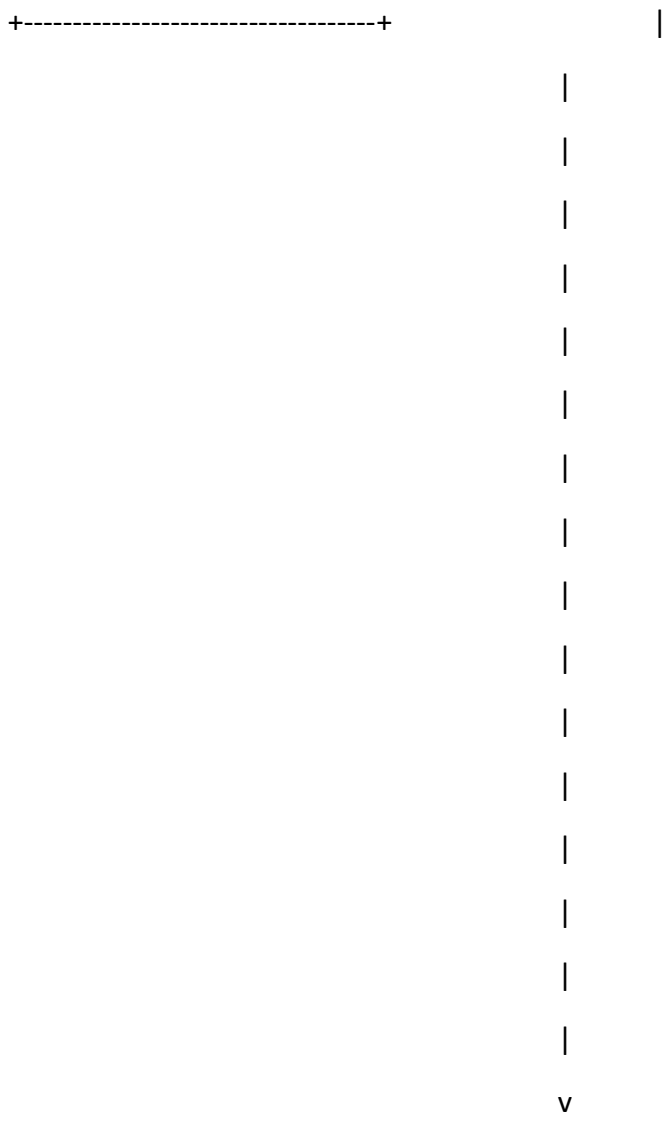
   k. The program terminates with a return code of 0.

Based on this description, we will now proceed to create a sequence diagram illustrating the flow of operations, including key steps such as video input processing, dynamic object detection, and overlay generation.

Here's the sequence diagram illustrating the flow of operations based on the provided code:

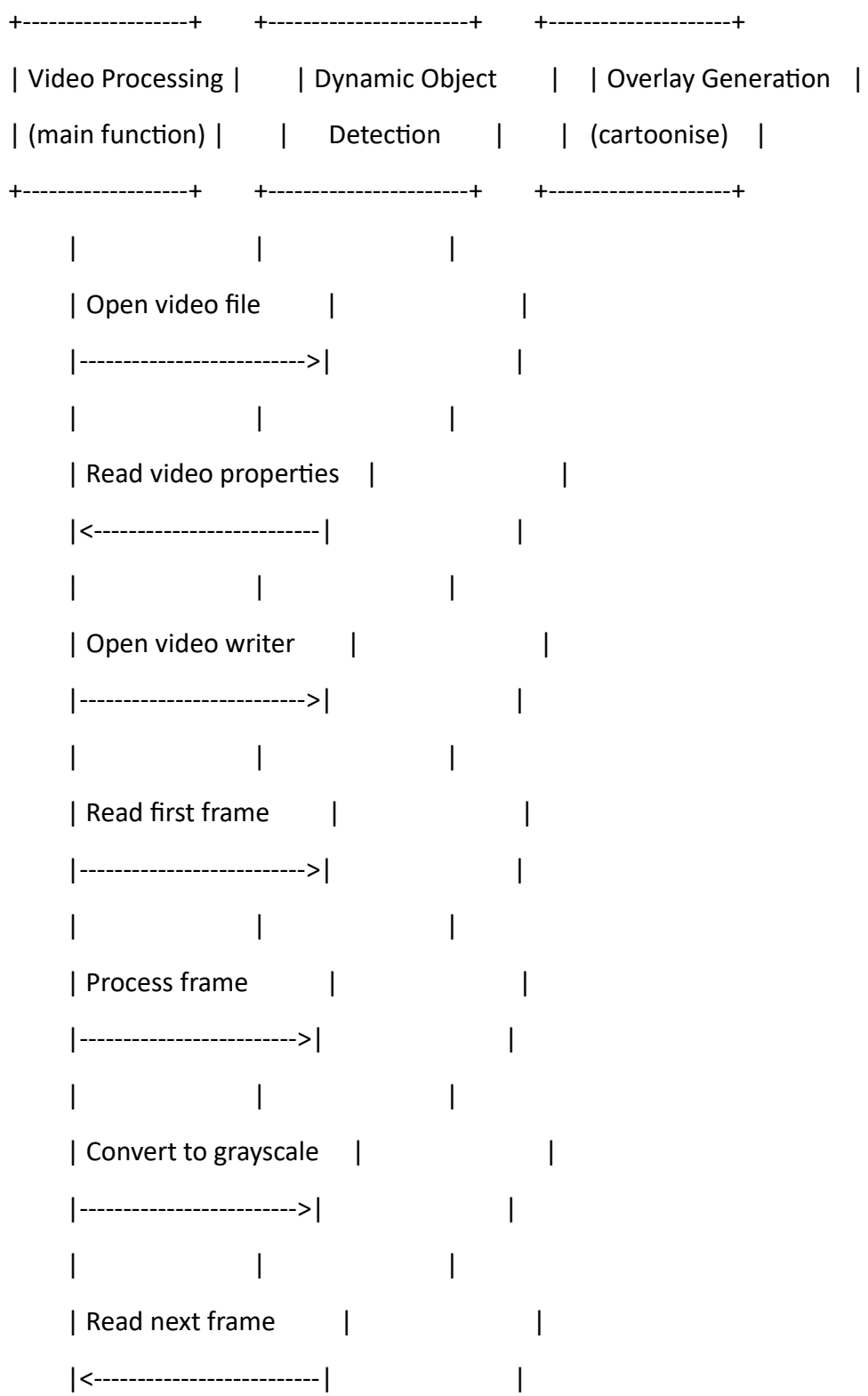


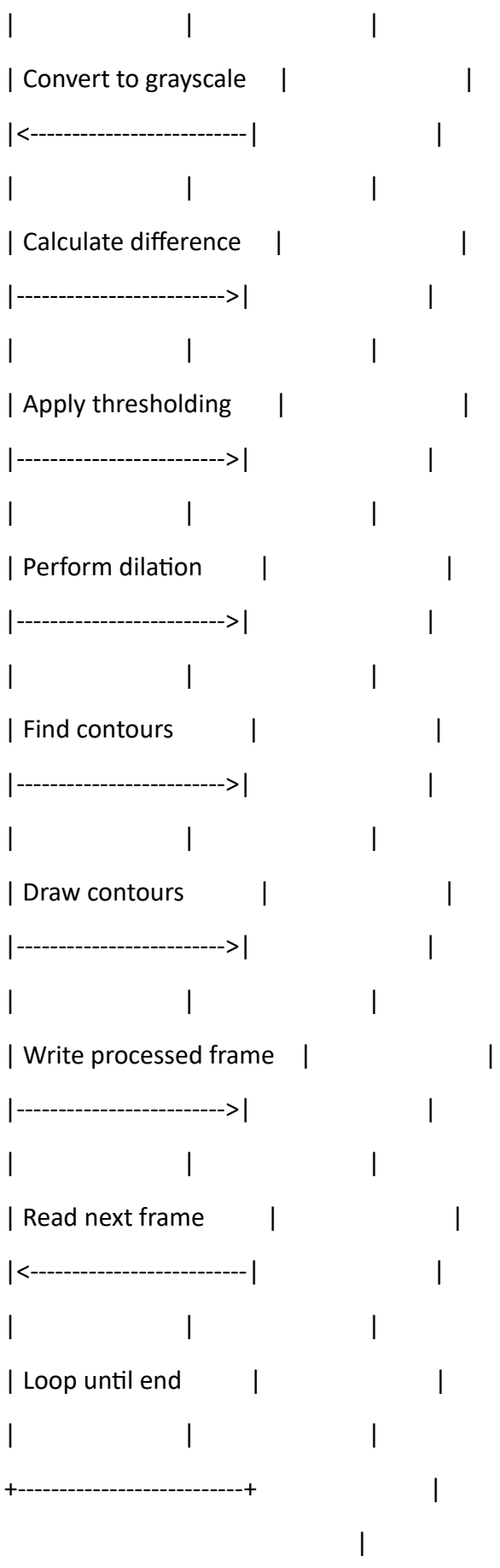

Convert frame to grayscale
Calculate absolute difference between consecutive frames
Apply thresholding
Perform dilation
Find contours
Draw contours on frame
Write processed frame to output video file
Read next frame
Loop until all frames processed



This diagram illustrates the flow of operations in the project, including video input processing, dynamic object detection, and overlay generation. Each box represents a distinct function or operation, and arrows depict the flow of control between them. The sequence starts with opening the video file and ends with processing all frames and writing the output video file.

The revised sequence diagram:




Convert to grayscale
Calculate difference
Apply thresholding
Perform dilation
Find contours
Draw contours
Write processed frame
Read next frame
Loop until end

This sequence diagram illustrates the flow of operations in the project, including video input processing, dynamic object detection, and overlay generation. Each box represents a distinct function or operation, and arrows depict the flow of control between them. The sequence starts with opening the video file and ends with processing all frames.