HPA Problem Statement Documentation

-By team: Translucent Seven

Translucent Seven Team Members:

Member 1:
Name: Harsh Milind Mhetre
Year of Study: 3rd Year B.E Computer Engineering
Institute: PES Modern College of Engineering, Pune
Email: harshmhetre99@gmail.com
Mobile No.: +91 7387519039

Member 2:
Name: Pradnya Raju Gaikwad
Year of Study: 3rd Year B.E Computer Engineering
Institute: PES Modern College of Engineering, Pune
Email: pradnya110503@gmail.com
Mobile No.: +91 8778976767

Problem Statement:

To detect the pedestrians in the provided video and outlining their visualisations

Libraries Used:

<iostream>: For basic coding structure in C++
<opencv2\opencv.hpp>: For computer vision tasks
<Windows.h>: For memory allocation tasks

Code Content:

This C++ code utilises the OpenCV library to detect pedestrians in a video stream using background subtraction and contour detection techniques. Let's break down the algorithm:

1. Include Libraries:
The code includes necessary header files from OpenCV, along with `<iostream>` for input/output operations and `<Windows.h>` for Windows-specific memory management functions.

2. Main Function:
The main function accepts command-line arguments for the video file path and base memory address.

3. Argument Validation:
It checks if the correct number of command-line arguments (video file path and base address) are provided. If not, it displays the correct usage and exits.

4. Memory Address Conversion:
The base memory address provided as a command-line argument is converted to a 64-bit unsigned integer and then to a `uintptr_t` type, which represents a pointer-sized unsigned integer type.

5. Memory Allocation:
Using the `VirtualAlloc` function from the Windows API, memory is allocated at the specified base address. The allocated memory region has a size of 16 MB (0x1000000 bytes) and is committed for both reservation and writing (using `MEM_RESERVE | MEM_COMMIT` flags).

6. Background Subtraction:

A background subtractor model (`cv::BackgroundSubtractorMOG2`) is created to separate foreground objects from the background in each frame of the video.

7. Video Capture:
The video file specified in the command-line argument is opened using `cv::VideoCapture`.

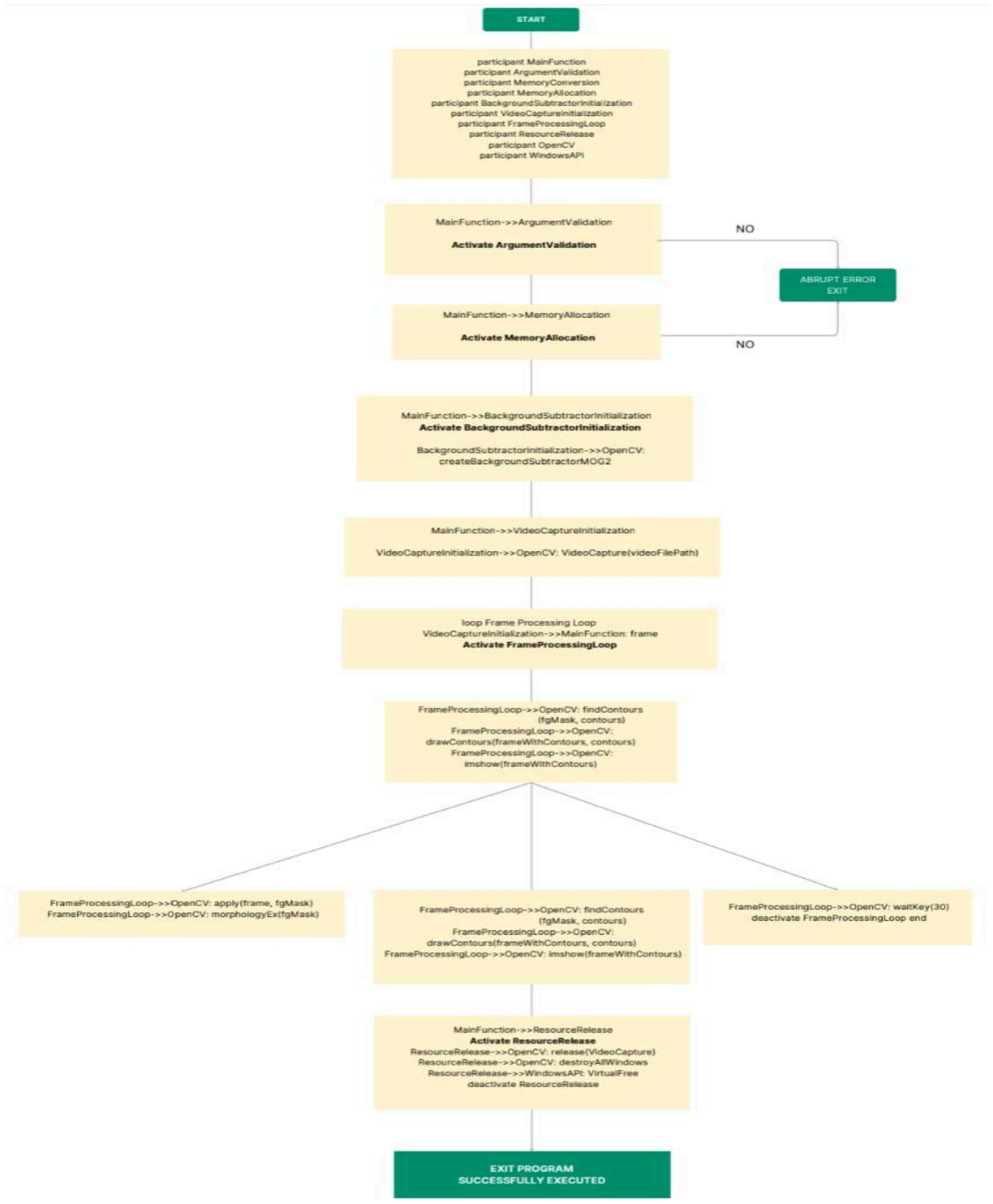
8. Frame Processing Loop:
The code reads each frame from the video file in a loop. For each frame:
- A copy of the original frame **(`frameWithContours`)** is made to draw pedestrian outlines later.
- Background subtraction is applied to the frame to extract the foreground mask (`fgMask`), which highlights potential pedestrian locations.
- Morphological operations (opening and closing) are performed on the foreground mask to remove noise and smooth the mask.
- Contours are detected in the processed foreground mask using `cv::findContours`.
- Outlines are drawn around the detected contours representing pedestrians on the `frameWithContours` image.
- The frame with pedestrian outlines is displayed in a window named "Video with Pedestrian Outlines".
- A delay of 30 milliseconds is introduced using `cv::waitKey` to control the frame rate of the video playback. The loop continues until the 'ESC' key (key code 27) is pressed.

9. Resource Release:
After processing all frames, the video capture is released, and all OpenCV windows are destroyed.

10. Memory Deallocation:
The allocated memory region is released using `VirtualFree` function to ensure proper cleanup.

Overall, this code demonstrates a simple pedestrian detection algorithm using background subtraction and contour detection techniques applied to a video stream using OpenCV.

Sequence Diagram:


START
participant MainFunction
participant ArgumentValidation
participant MemoryConversion
participant MemoryAllocation
participant BackgroundSubtractorInitialization
participant VideoCaptureInitialization
participant FrameProcessingLoop
participant ResourceRelease
participant OpenCV
participant WindowsAPI
MainFunction->>ArgumentValidation
Activate ArgumentValidation
NO
ABRUPT ERROR EXIT
MainFunction->>MemoryAllocation
Activate MemoryAllocation
NO
MainFunction->>BackgroundSubtractorInitialization
Activate BackgroundSubtractorInitialization
BackgroundSubtractorInitialization->>OpenCV: createBackgroundSubtractorMOG2
MainFunction->>VideoCaptureInitialization
VideoCaptureInitialization->>OpenCV: VideoCapture(videoFilePath)
loop Frame Processing Loop
VideoCaptureInitialization->>MainFunction: frame
Activate FrameProcessingLoop
FrameProcessingLoop->>OpenCV: findContours (fgMask, contours)
FrameProcessingLoop->>OpenCV: drawContours(frameWithContours, contours)
FrameProcessingLoop->>OpenCV: imshow(frameWithContours)
FrameProcessingLoop->>OpenCV: apply(frame, fgMask)
FrameProcessingLoop->>OpenCV: morphologyEx(fgMask)
FrameProcessingLoop->>OpenCV: findContours (fgMask, contours)
FrameProcessingLoop->>OpenCV: drawContours(frameWithContours, contours)
FrameProcessingLoop->>OpenCV: imshow(frameWithContours)
FrameProcessingLoop->>OpenCV: waitKey(30)
deactivate FrameProcessingLoop end
MainFunction->>ResourceRelease
Activate ResourceRelease
ResourceRelease->>OpenCV: release(VideoCapture)
ResourceRelease->>OpenCV: destroyAllWindows
ResourceRelease->>WindowsAPI: VirtualFree
deactivate ResourceRelease
EXIT PROGRAM
SUCCESSFULLY EXECUTED

Algorithm:

STEP 1 : Input Validation:
Check if the correct number of command-line arguments (video file path and base address) are provided.
If not, display the correct usage and exit.

STEP 2 :Memory Address Conversion:
Convert the base memory address provided as a command-line argument to a 64-bit unsigned integer and then to a uintptr_t type.

STEP 3: Memory Allocation:
Allocate memory at the specified base address using VirtualAlloc from the Windows API.
Check for successful allocation.
If allocation fails, display an error message and exit.

STEP 4 :Background Subtraction Initialization:
Create a background subtractor model (cv::BackgroundSubtractorMOG2) for foreground object detection.

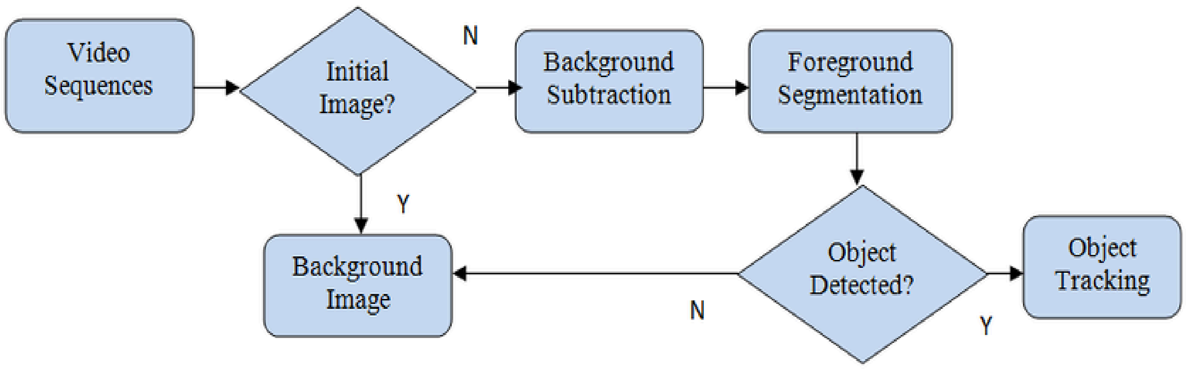
STEP 5 :Video Capture Initialization:
Open the video file specified in the command-line argument using cv::VideoCapture.
Check for successful opening of the video file.
If opening fails, display an error message and exit.

STEP 6 : Frame Processing Loop:
Read each frame from the video file in a loop until the end of the video or until the 'ESC' key is pressed.
For each frame:
-Apply background subtraction to extract the foreground mask.
-Perform morphological operations (opening and closing) on the foreground mask to remove noise and smooth the mask.
-Find contours in the processed foreground mask using cv::findContours.
-Draw outlines around detected contours representing pedestrians on a copy of the original frame.
-Display the frame with pedestrian outlines.
-Introduce a delay using cv::waitKey to control the frame rate of the video playback.

STEP 7 :Resource Release:
Release the video capture object.
Destroy all OpenCV windows.

STEP 8 :Memory Deallocation:
Free the allocated memory region using VirtualFree.

STEP 9 : End of Program:
Return 0 to indicate successful execution and termination of the program.



Design Choices:

To accomplish the given task we decided on an approach that is more lightweight to use and is more easy to understand.

We used the in-built sources provided by the OpenCV which allow us to perform object detection as well as the image processing.

We used a sequence of three methods to achieve Object Detection:

1. Background Subtraction
2. Morphological Operations
3. Contour Detection

We by studying through various materials found these three methods that could achieve the desired output and can be applied to real-time processing. These algorithms can later be fine-tuned to our required needs.

Background Subtraction:

Background subtraction is a fundamental technique used in computer vision and image processing to segment moving objects from the static background in a video sequence. It works by comparing each frame in a video stream with a background model to detect changes caused by moving objects. The background model represents the static elements of the scene, allowing foreground objects, such as pedestrians, vehicles, or other moving entities, to be isolated.

The process of background subtraction involves the following steps:
1. Initialization: Creating an initial background model from the first few frames of the video.
2. Updating: Adapting the background model over time to account for gradual changes in illumination or scene dynamics.
3. Foreground Detection: Subtracting each frame from the background model to identify regions where significant changes occur, indicating the presence of moving objects.
4. Post-processing: Applying morphological operations to refine the foreground mask and remove noise or artifacts.

Morphological Operations:

Morphological operations are image processing techniques used to manipulate the shape and structure of objects within an image. They are based on mathematical operations performed on the pixels of an image to achieve specific effects like noise reduction, edge detection, and shape enhancement. In the context of foreground segmentation in computer vision, morphological operations are often applied to binary images, such as foreground masks obtained from background subtraction.

Contour Detection:

Contour detection is the process of identifying and extracting the boundaries of objects within an image. In computer vision, contours are represented as sequences of points that define the shape of an object in the image. Contour detection algorithms analyze the pixel intensity gradients or binary image structures to identify regions with significant changes in intensity or color, which typically correspond to object boundaries.

The contour detection process involves the following steps:
1. Edge Detection: Detecting edges in the image using techniques like gradient-based methods, Canny edge detection, or thresholding.
2. Contour Initialization: Finding initial contours based on edge pixels or binary image structures.
3. Contour Tracing: Tracing the contours by following the connected edge pixels and forming closed loops around objects.

Algorithmic Approach:

1. The Program starts with including the necessary libraries, by using only OpenCV as a base library we reduce the heavy usage of the memory.
2. The Video Capture loads the video frame by frame, where the Background Subtractor from the OpenCV separates the frame to Background and Foreground where Background consists of non-moving static objects and the Foreground consists of any Moving objects found in the video.
3. Then Morphological Transformations are applied by involving the masks produced by the Subtractor for each moving object found in that particular frame, where the frame is recognized for Edge Detection and Dilution by analyzing the pixel distribution. This helps in understanding the boundaries of the image. It is also used to remove excess noise of the image.
4. Lastly, Contour Detection is used to recognize the masks provided by the Morphological transformations, where the objects are detected and their edges are taken into consideration. Through that knowledge it applies the required outlines over the edge by arranging the polylines to form the desired shape.
5. After the detection procedure is complete, the original background is loaded onto which the contours are overlaid to establish a successful visualisation.
6. The result is then shown by the OpenCV in a separate window with the detections overlaid.

Scope:

We thought of using such an approach as it is limited not only for people or vehicles but is applicable to any moving object, when we consider an Autonomous Driving or Parking Assistance; If we consider relative motion for a static object present can still be detected for if the vehicle itself is in motion. In addition we can align more models on to this particular algorithm that will work beside this for a much bigger scope. OpenCV itself uses a Dynamic Storage handler, but can be restricted to use a smaller percentage of Memory.

That particular fact has been proven where Computer Vision Tasks through OpenCV were executed via Raspberry PI.

We can further use different functions to make sure that memory utlization is optimal and not random. Functions like Mat.release(), or maybe not using cloning methods for frame copies. Loops can be thoroughly managed to not include any unrequired process to be recurring.

Execution:

The Zip File contains an C++ executable file and a source code for that program.

Before execution we must first ensure that you have the required packages. In our case we require the OpenCV.

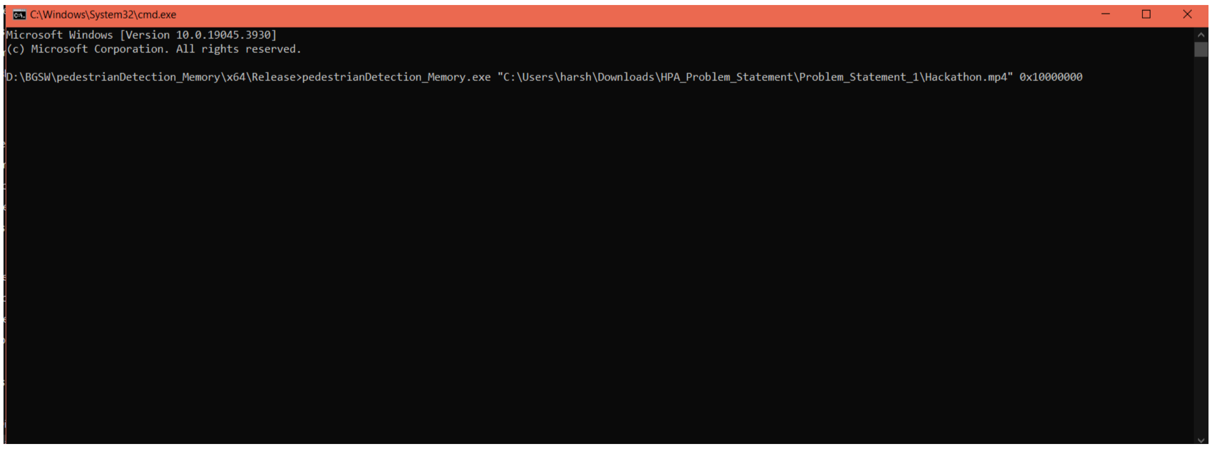
<opencv2/opencv.hpp>

Following are the steps to run the program:

1. The executable file is to be via command terminal by passing two arguments which are the video file input and the base address

   Syntax:

   Program_name.exe “video_file_path” baseAddress



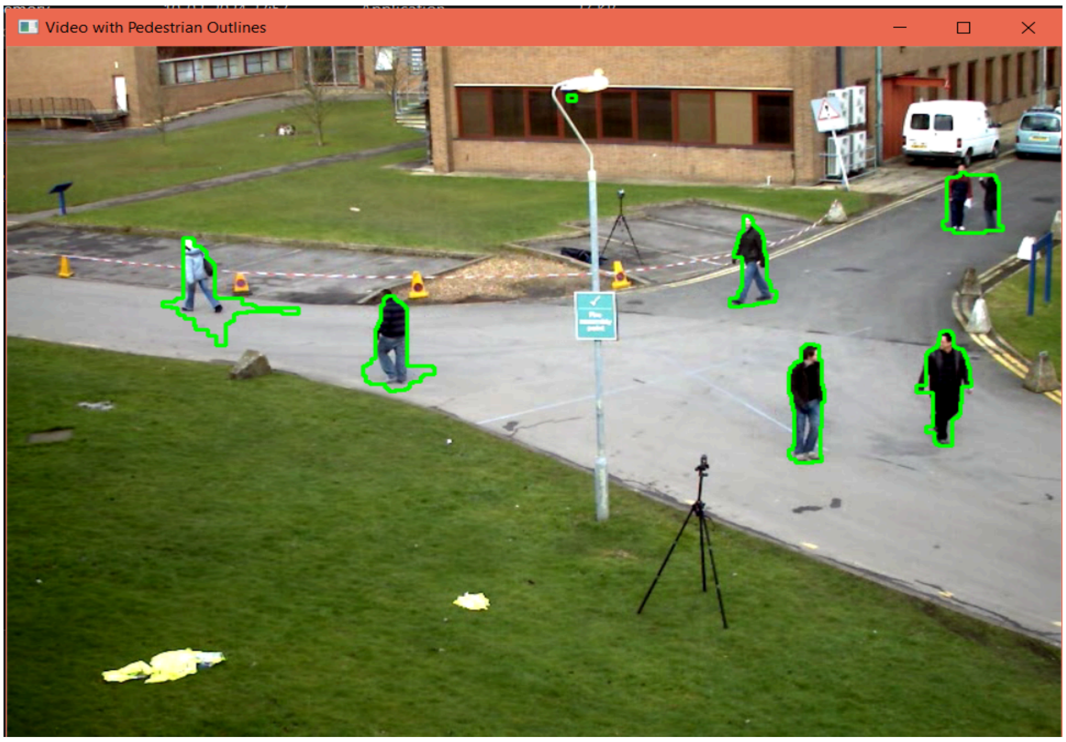
```
Microsoft Windows [Version 10.0.19045.3930]
(c) Microsoft Corporation. All rights reserved.

D:\BGSW\pedestrianDetection_Memory\x64\Release>pedestrianDetection_Memory.exe "C:\Users\harsh\Downloads\HPA_Problem_Statement\Problem_Statement_1\Hackathon.mp4" 0x10000000
```

2. The Video File Path must be enclosed in **“***Quotation Marks***”**
   For example:
   “C:/Users/harsh/Downloads/HPA_Problem_Statement/Problem_Statement_1/Hackathon.mp4”
   Also to ensure compatibility use ‘/’ rather than ‘\’ in the file path.

3. The Base address is to be passed right after the file path argument. If the base address that is being passed in is unavailable or invalid then an error message will be presented with the error code. In case of regular errors, then we can sort to a 32-bit base address and that will work.
   For example: 0x10000000

Output:


Video with Pedestrian Outlines