

**Dynamic Object Detection for Autonomous Driving**

**Team:Tech Troops CBI**

Sunilkumar S, III ECE, Sri Krishna College of Engineering and Technology,

Varon Atarrsh U, III ECE, Sri Krishna College of Engineering and Technology,

**1.Introduction:**

The problem of real-time dynamic object detection in autonomous driving is addressed through computer vision techniques. Leveraging OpenCV, our approach implements BackgroundSubtractorMOG2 for background subtraction, aiding in identifying moving objects. Our C++ solution processes video footage, highlighting dynamic objects with bounding boxes. With efficient memory management and x64 compatibility, our solution contributes to safer autonomous navigation.

**2.Approach:**

Our approach focuses on real-time dynamic object detection for autonomous driving systems. Upon receiving the video file input, we utilize computer vision techniques to process each frame in real-time. The video is captured using OpenCV's VideoCapture module, allowing us to extract individual frames. We then apply the BackgroundSubtractorMOG2 algorithm to perform background subtraction, isolating moving objects from the static background. This algorithm helps in identifying dynamically moving objects within the scene. Subsequently, contour detection is employed to extract the contours of these moving objects. We filter these contours based on their area to remove noise and retain only significant objects. Finally, we draw bounding boxes around the detected objects to visualize them in the video stream. This approach enables us to efficiently detect and highlight dynamically moving objects, such as vehicles and pedestrians, contributing to enhanced situational awareness for autonomous driving systems.

**2.1 Architecture**

The project aims to address the challenge of real-time dynamic object detection in autonomous driving scenarios using computer vision techniques. Leveraging OpenCV, the system implements BackgroundSubtractorMOG2 for background subtraction to identify moving objects. Through efficient memory management and compatibility with the x64 architecture, the solution contributes to safer autonomous navigation.

The architecture comprises several modules designed to facilitate real-time dynamic object detection:

Phase 1: Preparing:

Video Input Handling: The user is prompted to input the path to a video file, such as an MP4 file. This input path is used to initialize a cv::VideoCapture object, which allows the program to access and process the video frames.

Error Handling: If the specified video file fails to open, the program displays an error message indicating the failure and exits gracefully with an error code (-1).

Phase 2: Dynamic Memory Allocation:

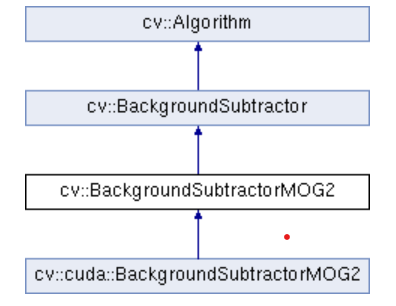
Buffer Allocation: The program dynamically allocates memory for a buffer to store the video frames. The buffer size is calculated based on the dimensions (width and height) of the video frames and assumes a 3-channel format (BGR) for color images.

User-Specified Base Address: The user is prompted to enter a memory address in hexadecimal format. This address serves as the base address for the allocated buffer, allowing for flexibility in memory management.

Memory Adjustment: To ensure proper memory access, the program calculates the offset from the user-specified base address and adjusts the buffer pointer accordingly.

Phase 3: Object Detection:

Background Subtraction: Utilizing the MOG2 algorithm, the program creates a background subtractor object (cv::BackgroundSubtractorMOG2) to detect moving objects within the video frames.



Frame Processing: The program iterates through each frame captured from the video input. It applies background subtraction to identify regions of interest (i.e., moving objects) and generates a binary mask representing these regions.

Object Detection and Visualization: Contours are extracted from the binary mask using cv::findContours, and rectangles are drawn around the detected objects based on predefined criteria (e.g., contour area threshold). This step visually highlights the moving objects within the video frames.

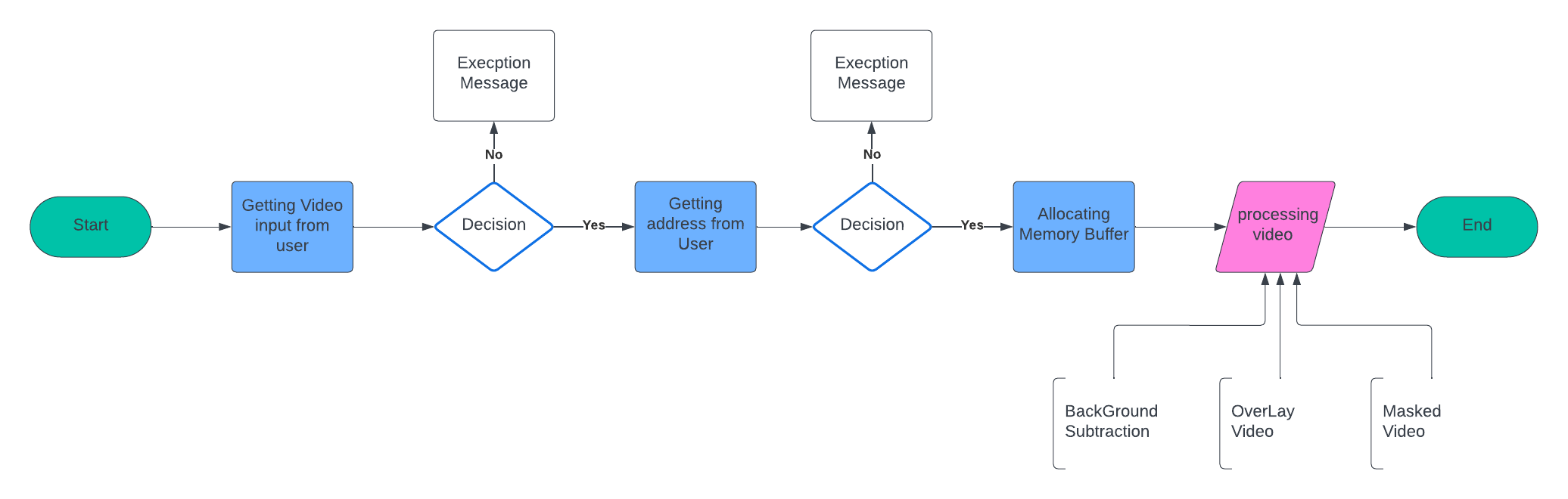
Interactive Display: The processed video frames, along with the corresponding binary masks, are displayed in separate OpenCV windows ("Frame" and "Mask") using cv::imshow. The program waits for a key press and exits the loop if the 'Esc' key is pressed.

Cleanup:

Resource Release: Upon completion of video processing or user interruption, the program releases system resources by closing the video capture (cap.release()) and destroying all OpenCV windows (cv::destroyAllWindows()).

Success Code: The program exits with a success code (0), indicating successful execution without encountering any errors.

**2.2 Sequence Diagram**

****

**3.Efficiency and Unique Approach in Dynamic Object Detection**

1. Real-time Processing: Our approach focuses on real-time processing of video data, enabling swift detection of dynamically moving objects as the video stream unfolds. This real-time capability ensures timely responses and enhances the system's ability to react to changing traffic conditions effectively.
2. Background Subtraction: By employing the BackgroundSubtractorMOG2 algorithm, our solution efficiently extracts moving objects by subtracting the background from each frame. This technique effectively isolates dynamic elements, minimizing computational overhead and enhancing detection accuracy.
3. Contour Detection and Filtering: We utilize contour detection to extract the shapes of moving objects from the foreground mask. By filtering contours based on area, our approach eliminates insignificant noise and retains only relevant objects, optimizing the detection process and improving the overall performance.
4. Bounding Box Visualization: Our solution visualizes detected objects using bounding boxes, providing clear and intuitive feedback to users. This visualization aids in understanding the spatial extent and location of moving objects, enhancing the system's interpretability and usability.
5. Memory Management and Compatibility: We ensure efficient memory management and compatibility with the x64 architecture, optimizing resource utilization and ensuring seamless integration with diverse hardware environments.

**4.Prerequisites or Components to Build and run our project:**

* + Visual Studio Code Community Edition
  + Visual Studio Build Tools 2022(version 17.8.3 or higher recommended)
  + CMake (version 3.28.2 or higher recommended)
  + OpenCV(version 4.9.0 (current) or latest version)
  + Visual Studio Extensions: C/C++ extension by Microsoft ,

CMake by twxs , CMake Tools By Microsoft

**5.Setup/Getting Started:**

Step 1: Download Libraries and CMake

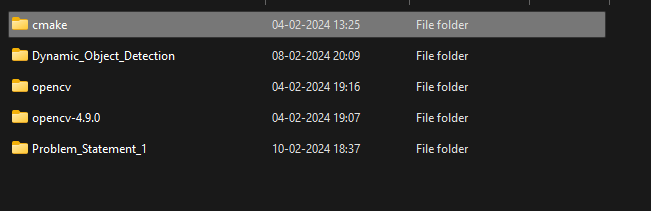
To download CMake go to: <https://cmake.org/download/> and Download X64 msi for windows / Any Operating System.

To Download OpenCV go to: <https://opencv.org/releases/> and Download version 4.9.0 or higher installer.

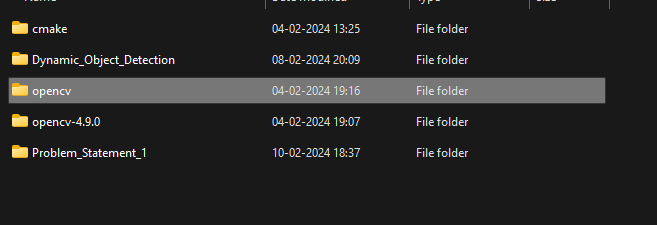
Step 2: Extract or Install the Downloaded Files create new Project Folder.

Click on CMake installer and enable the System path to all users and choose the Installation Path to project folder then install CMake in Your PC(Create a folder and put all files inside. Name of the folder may be your wish).

Inside The Project Folder:



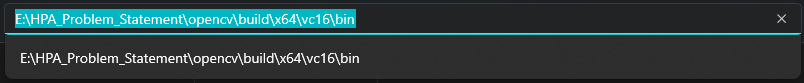
Install or Extract the OpenCV Library in the System inside the project folder:



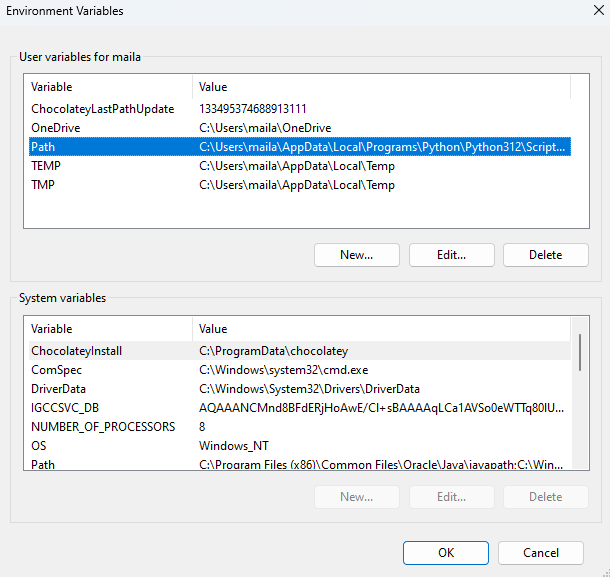
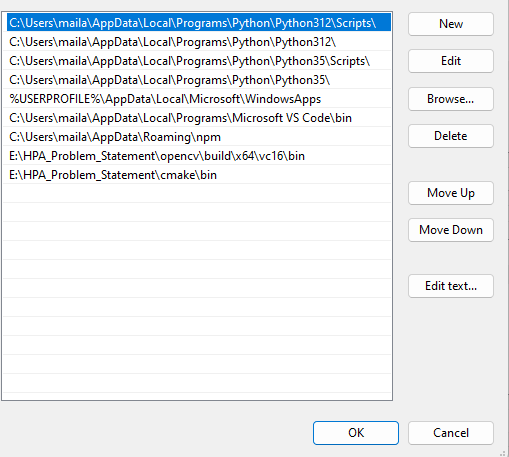
Step 3:Add path Variables for Seamless Integration.

Copy the bin folder’s path which is available opencv folder which is most Like this

”E:\HPA\_Problem\_Statement\opencv\build\x64\vc16\bin”.

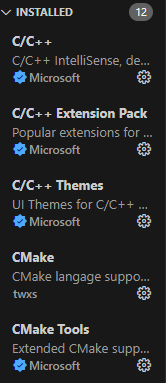


Add it to the System Path Variable to Seamless integration with other applications.



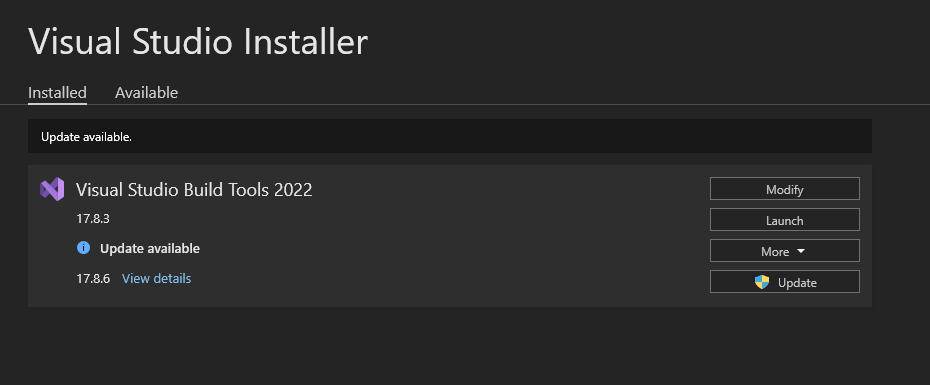
Step 4:Setting Up VS Code .

Install all the extensions in the Below image and restart your vs code:

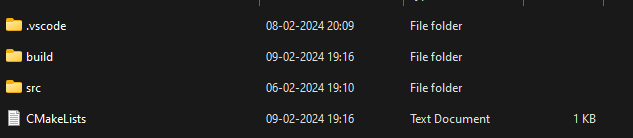


Step 5: Setting up Microsoft Visual Studio Build Tools

Install the Visual Studio Build Tools of the Below Shown version or higher.

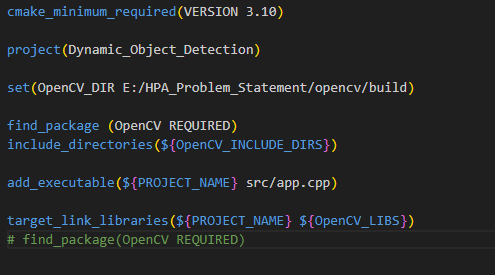


Step 6:Create Folder named Dynamic\_Object\_Detection inside project folder and create an file named CMakeLists.txt inside that folder.

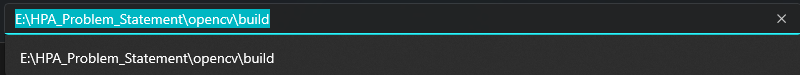


Create Source Folder(src) and Create an file named app.cpp inside src folder.

The CMakeLists.txt code Should be Like This:

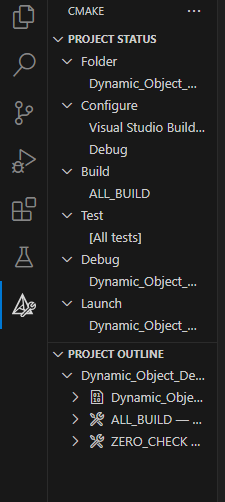


In the 5th line of code add the build folder path which is available opencv folder you created earlier (Yours may vary) for the OpenCV Directory access.

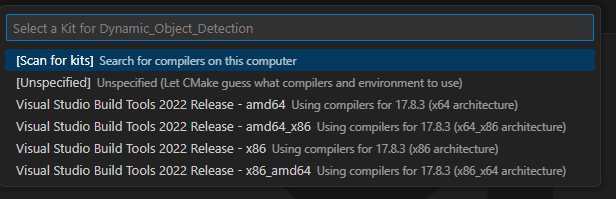


Step 7 Configuration : To configure Project go to CMake Tools in the side window

Click “Configure” and Select “Visual Studio Build Tools 2022 Release - x86\_amd64 Using compilers for 17.8.3 (x86 x64 architecture)” This Makes Our Code Compatible with x86 and x64 architecture.



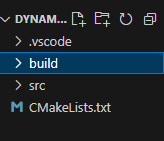
(Note: Please make sure you have Installed Microsoft Build Tools 2022 to Select the kit)



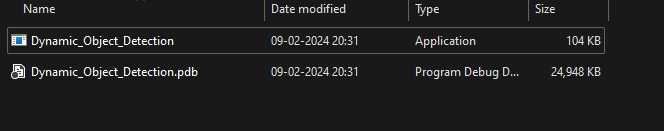
Click on Build to Build your Project



After Building completed You should be able to see the build folder inside your project

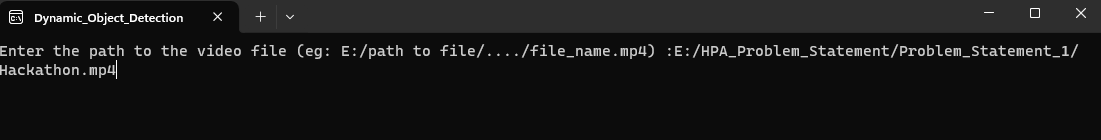


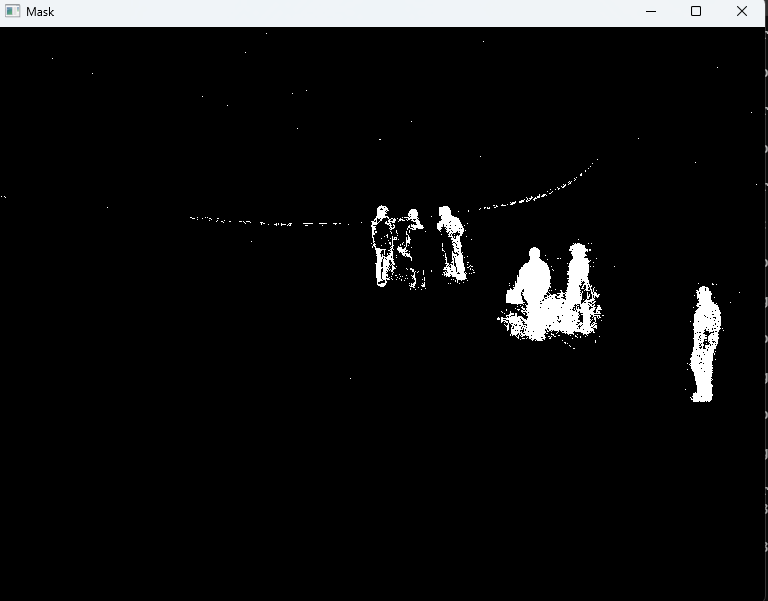
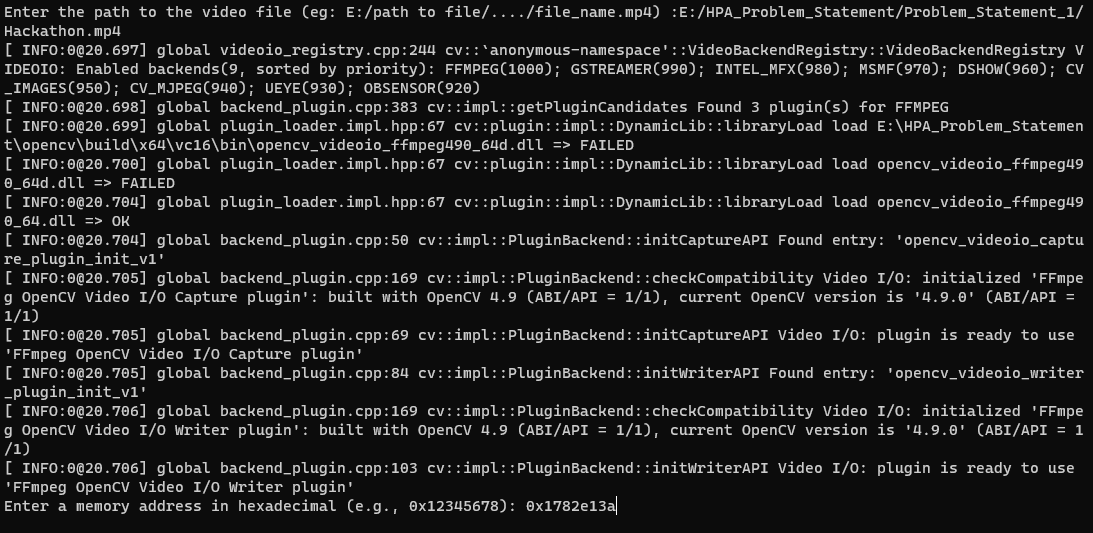
Inside Build>>Debug there is an executable(.exe) file of our project.



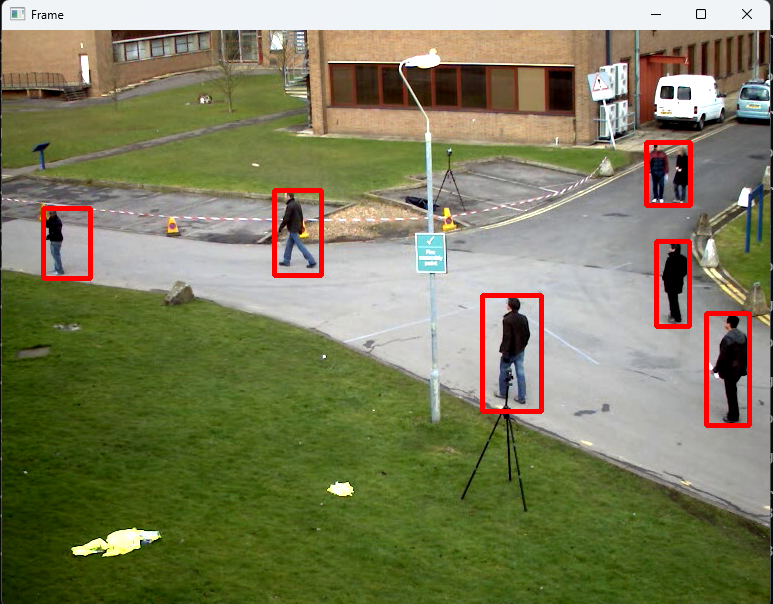
Step 8: Inside main.cpp there should be an program to do Dynamic Object Detection.

The program should take Video path and Buffer Memory address as input in command line interface and show overlay video as output



Mask Output (Gray scale):

Frame output:



**6.Results:**

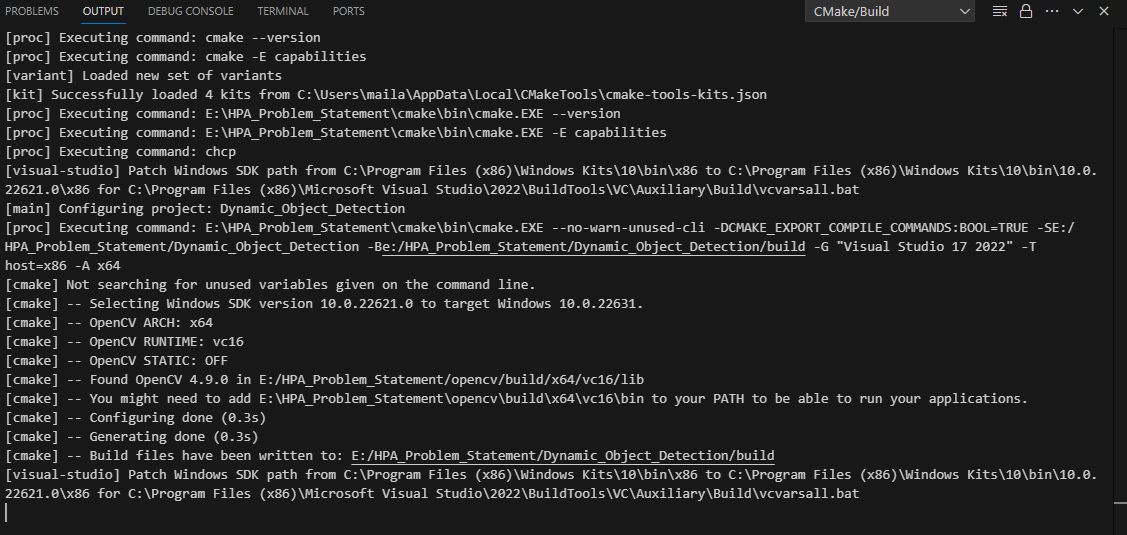
**6.1 C++ Project:**

The project is be implemented in C++.And it Utilizes appropriate libraries and frameworks for image processing and computer vision tasks. (refer in annexature section below)

**6.2 Executable Compatibility:**

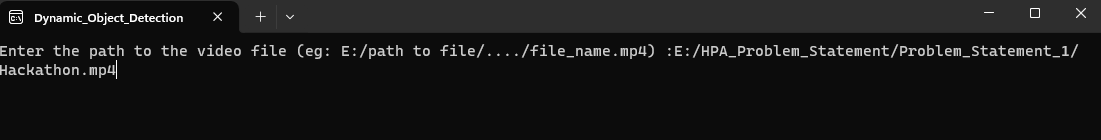
The executable or .exe file is compatible with the x64 architecture. Because it is built with “Visual Studio Build Tools 2022 Release - x86\_amd64 Using compilers for 17.8.3 (x86 x64 architecture)”. Done with Visual Studio Build Tools 2022.

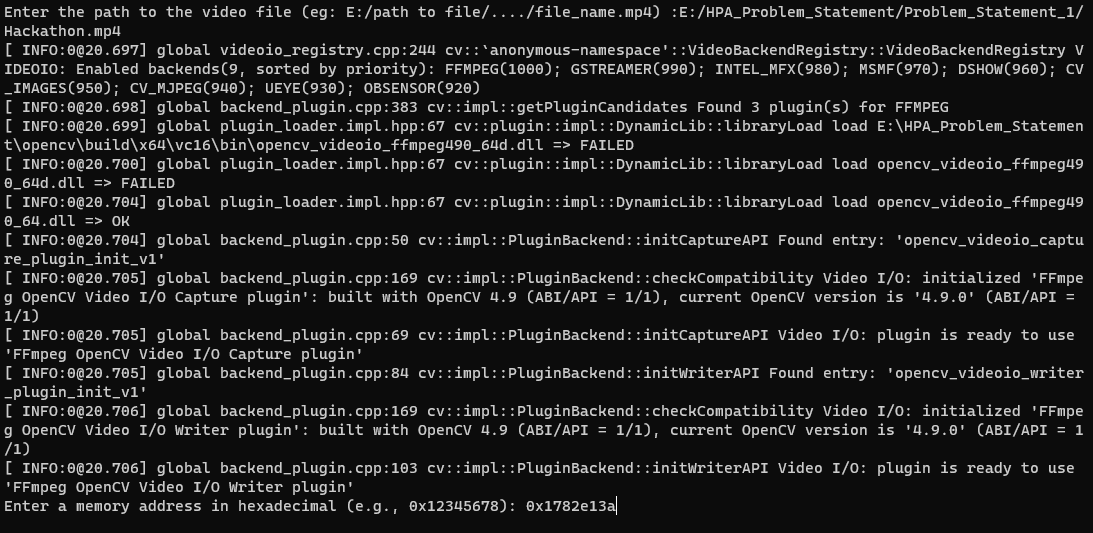
The architecture is specified with the option -T host=x86 -A x64. This means the project is set up to build for both x86 and x64 architectures.

Here is the Build Terminal log:  


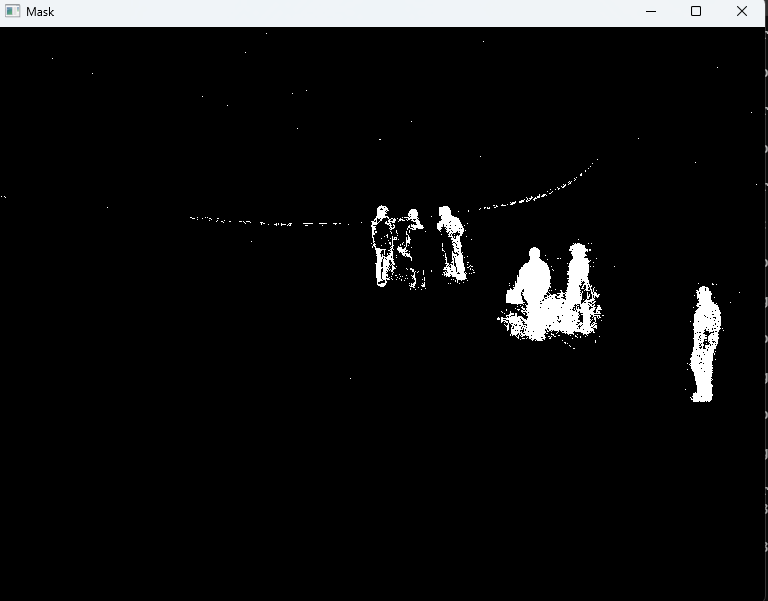
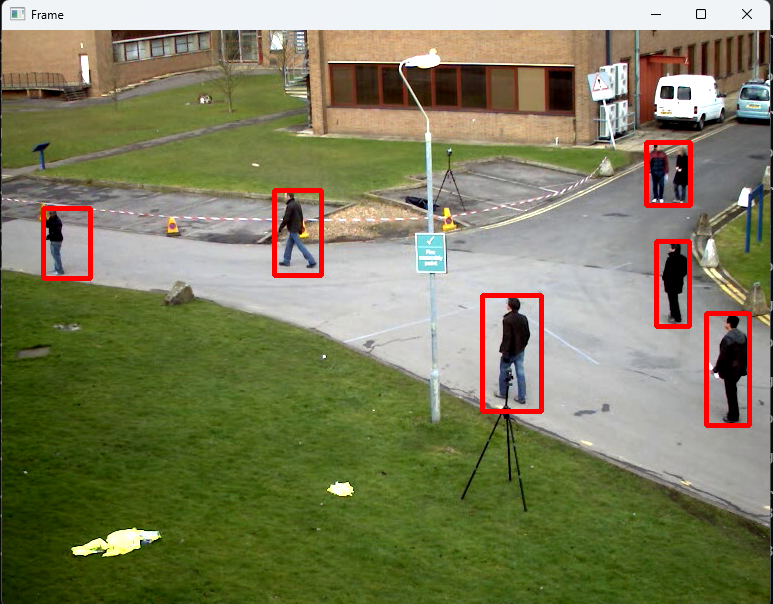
**6.3 Command-line Interface:**

The project is accepting two command-line arguments: the input video file path and the base address of memory.



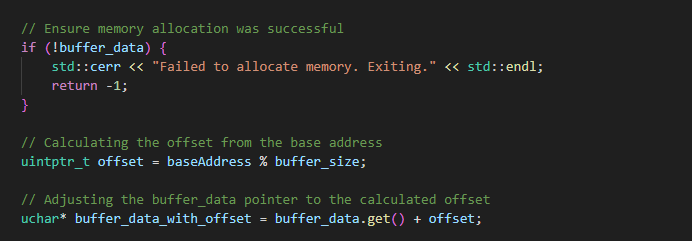
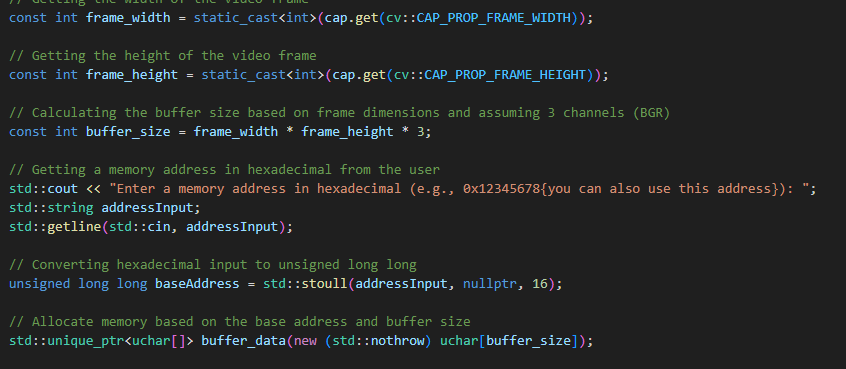


**6.4 Overlay Output:**

The output video displays the original video with an overlay indicating the detected dynamic objects. When the contour area is greater then 100 to ignore the very small movements. In addition we have displayed the Masking of gray scale of Dynamic Objects.

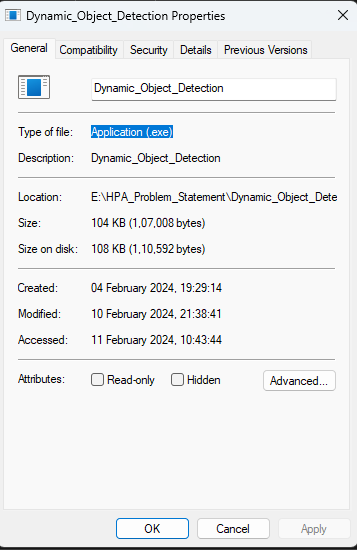
**6.5 Memory Management:**

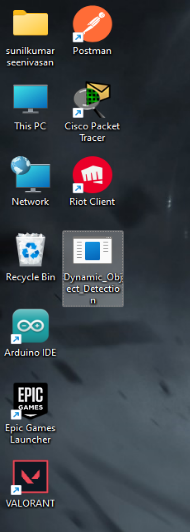
Efficiently manages memory allocations using the provided base address. This part of code allocates Buffer Memory based on the users input address.

****

**6.6 Executable :**

The Project can be executed using .exe file which is inside the Build>>Debug folder of Our Project it can be added to desktop as well





**7.Annexature:**

app.cpp:

#include <opencv2/opencv.hpp> // Including the OpenCV library after Installing

#include <iostream> // Including the input-output stream library

#include <memory> // Including the memory management library for smart pointers

#include <cstdlib> // Including the standard library for string to integer conversion

int main() {

//-------------------------------------------------------Phase 1: Preparing-----------------------------------------------------------------------------------//

// Declaring a variable to store the video file path

// And Prompting the user to enter the video file path

// Reading the video file path from the user as Input

std::string videoFilePath;

std::cout << "Enter the path to the video file (e.g., E:\path to file....\file\_name.mp4): ";

std::getline(std::cin, videoFilePath);

// Capturing the video from file

// Opening the video file specified by the user

cv::VideoCapture cap(videoFilePath);

// Ensuring the Video Path

// Check if the video file was successfully opened

// Display an error message if the video file cannot be opened

// Return an error code indicating failure

if (!cap.isOpened()) {

std::cerr << "Error opening video file" << std::endl;

return -1;

}

//---------------------------------------------------Phase 2: Dynamic Memory Allocation--------------------------------------------------------//

// Getting the width of the video frame

const int frame\_width = static\_cast<int>(cap.get(cv::CAP\_PROP\_FRAME\_WIDTH));

// Getting the height of the video frame

const int frame\_height = static\_cast<int>(cap.get(cv::CAP\_PROP\_FRAME\_HEIGHT));

// Calculating the buffer size based on frame dimensions and assuming 3 channels (BGR)

const int buffer\_size = frame\_width \* frame\_height \* 3;

// Getting a memory address in hexadecimal from the user

std::cout << "Enter a memory address in hexadecimal (e.g., 0x12345678{you can also use this address}): ";

std::string addressInput;

std::getline(std::cin, addressInput);

// Converting hexadecimal input to unsigned long long

unsigned long long baseAddress = std::stoull(addressInput, nullptr, 16);

// Allocate memory based on the base address and buffer size

std::unique\_ptr<uchar[]> buffer\_data(new (std::nothrow) uchar[buffer\_size]);

// Ensure memory allocation was successful

if (!buffer\_data) {

std::cerr << "Failed to allocate memory. Exiting." << std::endl;

return -1;

}

// Calculating the offset from the base address

uintptr\_t offset = baseAddress % buffer\_size;

// Adjusting the buffer\_data pointer to the calculated offset

uchar\* buffer\_data\_with\_offset = buffer\_data.get() + offset;

//--------------------------------------------------------------Phase 3: Object Detection------------------------------------------------//

// Creating a background subtractor for object detection

// This line initializes a background subtractor object using the MOG2 algorithm for motion detection.

cv::Ptr<cv::BackgroundSubtractorMOG2> object\_detector = cv::createBackgroundSubtractorMOG2(100, 25, false);

// Loop infinitely until user interrupts

/\*Reads each frame from the video capture in a loop.

Performs additional processing on the video frame (e.g., background subtraction, object detection).

Displays the processed frame using cv::imshow.

Waits for a key press, and exits the loop if the 'Esc' key is pressed.\*/

while (true) {

// Read a frame from the video capture

cv::Mat frame;

bool ret = cap.read(frame);

if (!ret) break;

// Creating a mask for object detection

cv::Mat mask;

// Applying background subtraction to detect moving objects

object\_detector->apply(frame, mask);

// Threshold the mask

cv::threshold(mask, mask, 254, 255, cv::THRESH\_BINARY);

//-------------------------------------Overlay On Moving Object-------------------------------------//

// Finding contours and draw rectangles around moving objects

// Creating a vector to store contours

std::vector<std::vector<cv::Point>> contours;

// Finding contours in the mask

cv::findContours(mask, contours, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);

// Iterate over each contour

for (const auto& cnt : contours) {

// Calculating the area of the contour

double area = cv::contourArea(cnt);

// Checking if contour area is above a threshold

if (area > 100) {

// Getting the bounding box of the contour

cv::Rect bounding\_box = cv::boundingRect(cnt);

// Drawing a rectangle

cv::rectangle(frame, bounding\_box, cv::Scalar(0, 0, 255), 3);

}

}

// Display the processed frame

cv::imshow("Frame", frame);

// Display the processed mask

cv::imshow("Mask", mask);

// Wait for a key press in the end of the video

// Check if ESC key is pressed

// Break the loop if ESC key is pressed

int key = cv::waitKey(100);

if (key == 27) {

break;

}

}

// Release video capture and close windows

cap.release();

cv::destroyAllWindows();

// Return success code indicating successful execution

return 0;

}

